Classification Project (Fabio Costantino)

My role in the Company

As a Content Analyst in the Localization Team, I am responsible for migrating data in and out of our Content Management System called STEP. These are mostly text data - a.k.a. content, which are then published daily in our website for all customers to consume.

As a Localization team, we mainly deal with translated text in 12 different languages, therefore also managing the flow of translation files in and out of the company. In order to translate our content, I have daily contact with external translation agencies and manage the use of a Translation Management System called XTM, which helps organize all files and workflows in a single tool.

Besides, we also provide our manager(s) and the rest of the team with reports of many kind: from tailored collection of data coming from different sources, to reports on products (or families of products) performance.

Some of the tools I use on a daily/weekly basis to perform my job are: **STEP Content Management System**, **XTM Translation Management System**, **Microsoft SQL Server Management Studio**, **Report Builder** from **Adobe Analytics**, **DOMO** as a BI dashboard tool, **Office package**, **Jupyter Notebook** (with **Python**).

Brief

Create an algorithm that helps define enrichment priorities in foreign markets.

Business Problem

Since our company sells products in markets that are very different from each other, and where consumers (being them other businesses or private customers) behaves differently from one another, the need arises constantly to monitor the differences in purchasing behaviour and to adapt strategies at a local level. In the Content department, the revision of relevant webpages in local language is of paramount importance to capture the interest of the potential customer, leading to conversion.

Current Process

The current process in our company is very much UK-centric, and until a few years ago most markets would simply follow the decision of the UK Content team based on data collected solely within the UK. However, with time this approach was revealed to be sub-optimal and the Localization team has been preparing ad-hoc reports for single markets using Adobe Analytics data.

Current issues

The main issue with this approach seems to be that the same report and the same variables have been applied to all markets – which basically simply shifted a UK-centric solution to a one-size-fits-all solution, which is rapidly showing inefficiencies and discrepancies with local produced reports from local major stakeholders.

Proposed solution

The pilot project that I present in this paper has the goal of gathering more information than the usual 2 or 3 variables from the Adobe suite. To this extent, I gathered data from different departments and linked them to obtain a dataset with a wide range of information. Of course, this is by no means an exhaustive explanation of all possible variables, but an experiment reduced in scope for the purpose of testing this solution. I intend to use 3 algorithms that will classify the dataset based on the quantity sold in the past 12 months for the Japanese market. Once the labels have been applied, I will deploy the most successful algorithm (see Assumptions for definition) to a dataset for the Chinese market, which is similar for many aspects (see Assumptions).

Pros and Cons

In my opinion, the strength of this approach lies in the comparison of local markets that are similar, and not only with the UK local market. By further experimenting with this approach and combining similar markets, it should be also possible to create a content strategy for markets that we haven't yet entered, allowing to forecast costs and effort for expansion. On the other hand, one can also argue that no market is completely similar to another, and competition might depict a very different landscape for any two countries. Moreover, the choice of variables in the dataset certainly influences greatly and perhaps skews the outcomes.

Assumptions

1) *Most successful algorithm*: each model will be tested with a **precision-recall curve**, and all models will be compared visually by means of a **Receiving Operating Characteristics (ROC) curve** and mathematically by means of an **Area Under the Curve (AUC)** value. Success will therefore be defined as the best performance in both evaluations.

2) *Japanese and Chinese markets being similar*: this assumption has been corroborated by Business Users with deeper domain knowledge than myself on both markets.

3) This analysis is performed on **families of products**, therefore the data gathered are aggregated and not reflective of every single product available. A more granular analysis is suggested to be performed on single products, but this is computationally extremely expensive and out of the scope of the present project.

4) This analysis concentrates on the **products stored locally** in the market analyzed. This allows me to simplify my task, since products not stored locally would entail costs and constraints, the data for which I don't have access to.

0. Importing libraries

First of all I need to import all relevant libraries to this analysis.

In [1]:
```
# Basic dataset manipulation
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns
import plotly.express as px

# Preparation of dataset for modelling
from sklearn.model_selection import train_test_split

# Modelling random forest and logistic regression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Evaluation and results visualization
from sklearn import tree, metrics
from sklearn.metrics import confusion_matrix

# Estimation of information gain
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc

# Suppression of warnings
import warnings; warnings.simplefilter('ignore')
```

```
C:\Users\E0658269\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\tools\_testing.py:19: FutureWarning: pandas.util.testing is deprecated.
Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1. Gathering and transforming data

I needed to gather data from different sources and different departments. The contribution of each department will be briefly explained after the import of the complete dataset, using the `read_excel()` function contained in the Pandas library.

In [2]:
```
# Importing the dataset
data_JP = pd.read_excel('data_JP.xlsx')

# Checking that the import is successfull
data_JP.tail()
```

Out[2]:

	PSF	Technology	Total_no_products	Enrichment
3263	PSF_981843	TCFM	20	0
3264	PSF_990951	A&C	41	0
3265	PSF_990955	TCFM	40	0
3266	PSF_990959	TCFM	17	0
3267	PSF_999999	A&C	1392	1

At the time of writing, the UK Content team is divided into 3 technology sub-teams, i.e. **TCFM**, **A&C** and **BLE**, each of which is responsible for a certain number of products families. I created a list of all families in the system divided by technology and sent it to the UK team, asking them to double check that the information was correctly gathered.

The total number of items in each product family was pulled from SQL with the following code involving the complete list of family IDs (contained in the reference file called `AAA_Fabio.txt`) and a table containing the taxonomical structure of the entire website (`MP_tbl_PSHier`).

```
SELECT b.prod, COUNT(a.[Stock_Number])
FROM MP_tbl_PSHier a
inner join AAA_Fabio b
on a.Family_ID = b.prod
GROUP BY b.prod
```

Finally, I gathered the complete list of enriched products for the Japanese market from my colleagues of the **Localization team**, which gave me the chance to label my data in the `Enrichment` column as enriched (`1`) or not enriched (`0`).

The **Supply Chain** team provided me with a second file in **Microsoft Access**, that adds information on the net sales for each category and the quantity of products sold in each family. Before using these data coming from sensitive datasets, I discussed and received clearance from my contact in the Supply Chain department to actually use this information for analysis. Clearly asking for permission to include these data ensured that I don't break any internal protocol for information sharing.

I will now import this second file.

In [3]:
```
# Importing the second file
stock_JP = pd.read_excel('Japan_final.xlsx')

# Checking that the import was successfull
stock_JP.head()
```

Out[3]:

	PSF	Article	Category Net Sales	Quantity Sold
0	PSF_438292	1000150	[redacted]	[redacted]
1	PSF_430991	1000166	[redacted]	[redacted]
2	PSF_430915	1000655	[redacted]	[redacted]
3	PSF_433056	1001076	[redacted]	[redacted]
4	PSF_433056	1001149	[redacted]	[redacted]

To properly transform this dataset, I will need to specify that the column `Article` contains only strings. If I don't do so, further down the line in my analysis, these numbers will be read as integers and summed up accidentally.

In [4]:
```
# First I check the data types for all columns
stock_JP.dtypes
```

Out[4]:
```
PSF                   object
Article                int64
Category Net Sales    float64
Quantity Sold          int64
dtype: object
```

In [5]:
```
# Now I change the data type for 'Article' into string
stock_JP['Article'] = stock_JP['Article'].astype(str)

# Checking for correct implementation
stock_JP.dtypes
```

Out[5]:
```
PSF                   object
Article               object
Category Net Sales    float64
Quantity Sold           int64
dtype: object
```

There are still a few transformation I want to perform on these data. As the set is presented, right now we have many family IDs (called PSF) repeated for each product stored locally. I want to regroup these IDs and count the total summation of unique article numbers, quantity sold and net sales in each family for all locally stocked products.

In [6]:
```
# Aggregating the dataset by PSF ID and performing sum of relevant columns
stock_JP_grouped = stock_JP.groupby('PSF',as_index=True).agg({'Article': ['cou
nt'],
                                                              'Category Net Sa
les' : ['sum'],
                                                              'Quantity Sold'
: ['sum']})

# Checking for correct implementation
stock_JP_grouped.head()
```

Out[6]:

	Article	Category Net Sales	Quantity Sold
	count	sum	sum
PSF			
PSF_1014155	125	[redacted]	[redacted]
PSF_1022313	31	[redacted]	[redacted]
PSF_1069219	3	[redacted]	[redacted]
PSF_1073180	4	[redacted]	[redacted]
PSF_1073181	3	[redacted]	[redacted]

Now that the data are in the form I wanted, I can merge the two dataset to obtain a new one with all the information.

In [7]:
```
# Merging dataset on PSF IDs as unique identifier
merged_data_JP = pd.merge(data_JP, stock_JP_grouped, on = 'PSF')

# Checking the correct creation of the dataset
merged_data_JP.head()
```

Out[7]:

	PSF	Technology	Total_no_products	Enrichment	(Article, count)	(Category Net Sales, sum)	(Quantity Sold, sum)
0	PSF_432042	TCFM	1141	1	158	[redacted]	[redacted]
1	PSF_437181	A&C	404	1	23	[redacted]	[redacted]
2	PSF_437185	A&C	164	1	3	[redacted]	[redacted]
3	PSF_409441	TCFM	414	1	31	[redacted]	[redacted]
4	PSF_409464	TCFM	152	1	6	[redacted]	[redacted]

Finally, I want to calculate the ratio of products that are stored locally for each family. This will give me the opportunity to understand whether the Content team should take into account storage location in analyzing their priorities.

The calculation is performed as follows:

$$LocalStockRate = \frac{LocalStock}{TotalStock}$$

In [8]:
```
# Calculating the local stock rate
merged_data_JP['Local_stock_rate'] = (merged_data_JP.iloc[:,4] / merged_data_J
P.iloc[:,2])*100

# Checking for correct implementation
merged_data_JP.head()
```

Out[8]:

	PSF	Technology	Total_no_products	Enrichment	(Article, count)	(Category Net Sales, sum)	(Quantity Sold, sum)	Loca
0	PSF_432042	TCFM	1141	1	158	[redacted]	[redacted]	
1	PSF_437181	A&C	404	1	23	[redacted]	[redacted]	
2	PSF_437185	A&C	164	1	3	[redacted]	[redacted]	
3	PSF_409441	TCFM	414	1	31	[redacted]	[redacted]	
4	PSF_409464	TCFM	152	1	6	[redacted]	[redacted]	

As a very last (and cosmetic) transformation, I will rearrange and rename some of the columns to have a clearer view.

In [9]:
```
# Rearranging the columns
final_data_JP = merged_data_JP.iloc[:,[0,1,7,5,6,3]]

# Renaming appropriately some of the columns
final_data_JP.rename(columns={final_data_JP.columns[3]: "Net_sales", final_dat
a_JP.columns[4]: "Qty_sold"}, inplace = True)

# Checking the result
final_data_JP.head()
```

Out[9]:

	PSF	Technology	Local_stock_rate	Net_sales	Qty_sold	Enrichment
0	PSF_432042	TCFM	[redacted]	[redacted]	[redacted]	1
1	PSF_437181	A&C	[redacted]	[redacted]	[redacted]	1
2	PSF_437185	A&C	[redacted]	[redacted]	[redacted]	1
3	PSF_409441	TCFM	[redacted]	[redacted]	[redacted]	1
4	PSF_409464	TCFM	[redacted]	[redacted]	[redacted]	1

In [10]:
```
# A very last check to ensure that no null data was accidentally introduced in
the dataset
final_data_JP.info()
```

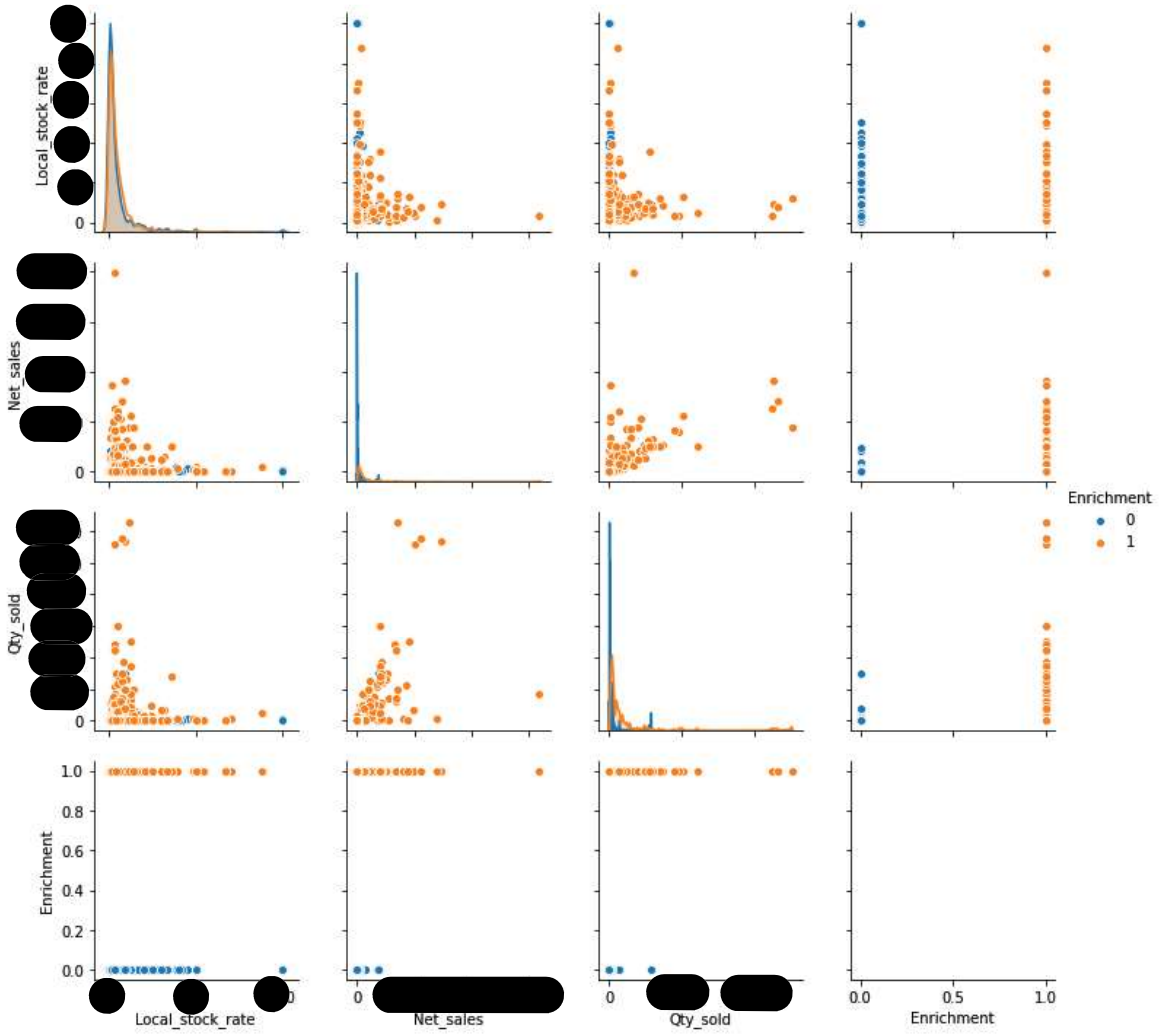
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1186 entries, 0 to 1185
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   PSF               1186 non-null   object
 1   Technology        1186 non-null   object
 2   Local_stock_rate  1186 non-null   float64
 3   Net_sales         1186 non-null   float64
 4   Qty_sold          1186 non-null   int64
 5   Enrichment        1186 non-null   int64
dtypes: float64(2), int64(2), object(2)
memory usage: 64.9+ KB
```

2. Visualizing data

The following visualizations will allow me to explore the dataset and understand it better.

In [11]:
```
# Creating a pairplot that emphasizes enriched and not enriched products
sns.pairplot(final_data_JP, hue='Enrichment')
```

Out[11]: <seaborn.axisgrid.PairGrid at 0x1c284b4aeb8>

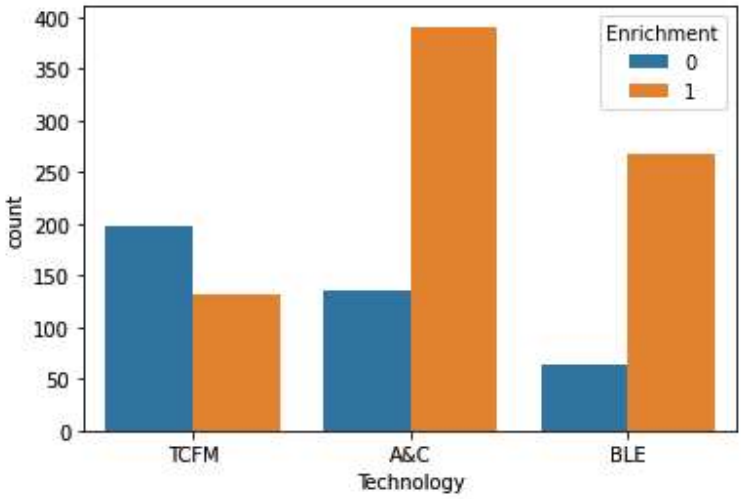


Looking at this graph, I can observe that a large number of products that didn't sell well (see plots for **net sales** and **quantity sold**) are not enriched. This is clearly not a surprise, rather a confirmation that taking care of the content will pay off.

Also, it seems that the quantity sold and net sales are not strongly correlated with the products being stocked locally. This is also not surprising, given that customers don't pay delivery costs.

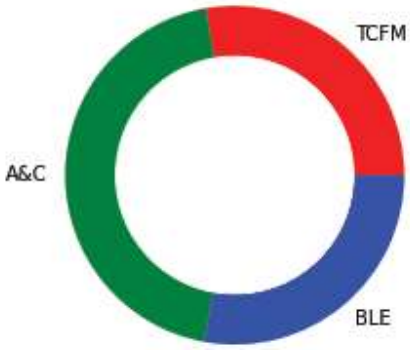
In [12]:
```
# Exploring the results divided by technology visually
sns.countplot(x='Technology', data=final_data_JP, hue='Enrichment')
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c284609160>
```



This graph seem to suggest that the **TCFM** team is a bit behind its work on content, therefore I decided to see if there is any discrepancy in the number of families assigned to each technology.

In [13]:
```
# Giving the necessary information to create the visualization
names='TCFM', 'A&C', 'BLE'

# Sizes are calculate as the amount of PSF IDs pertaining to a certain technol
ogy
size=[final_data_JP[final_data_JP['Technology'] == 'TCFM'].count().PSF,
      final_data_JP[final_data_JP['Technology'] == 'A&C'].count().PSF,
      final_data_JP[final_data_JP['Technology'] == 'BLE'].count().PSF]

# Creating a circle for the visualization
my_circle = plt.Circle( (0,0), 0.7, color='white')

# Giving colors and showing the plot
plt.pie(size, labels=names, colors=['red','green','blue'])
p=plt.gcf()
p.gca().add_artist(my_circle)
plt.show()
```

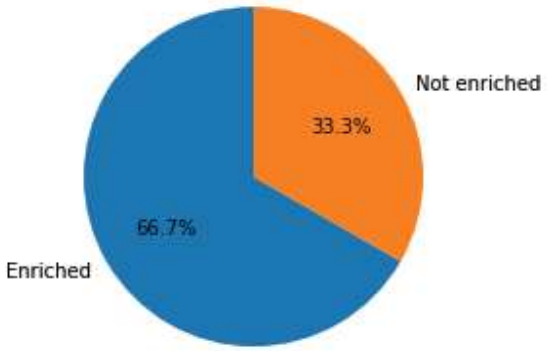


Apparently the total number of families is roughly equally distributed.

However, upon investigation in our systems, I noticed that the number of products in certain families vary widely and the TCFM team actually deals with a larger number of single products. The difference is therefore explained by cross-reference of the data and this information can be ignored for the moment.

Another information I want to obtain is the percentages of product enriched and not enriched.

In [14]:
```
# To obtain a pie chart, first I calculate how many families of products in to
tal have been enriched or not enriched
slices = [final_data_JP[final_data_JP['Enrichment'] == 1].count().PSF,
          final_data_JP[final_data_JP['Enrichment'] == 0].count().PSF]

# Then I select the labels I want for the two sections of the pie
enrichment = ['Enriched', 'Not enriched']

# Finally I select the correct function in matplotlib and launch the visualiza
tion
plt.pie(slices, labels=enrichment, startangle=90, autopct='%.1f%%')
plt.show()
```



Finally, I want to explore visually the correlation among the different columns, using the `heatmap()` function.

In [15]:
```
# First, I create a one-hot encoded version of the dataset (exluding the PSF I
Ds which are unique values)
corr_data = pd.get_dummies(final_data_JP.set_index('PSF'))

# Then I create a correlation matrix
corr_matrix = corr_data.corr()

# Finally I visualize the correlation matrix with the heatmap
plt.figure(figsize=(20, 10))
sns.heatmap(corr_matrix, annot=True)
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c283f27ba8>
```

The products that are not enriched appear to have a slight negative correlation with the net sales and quantity sold, whereas the local stock rate seem to be quite ineffective to explain our data.

3. Preparing the dataset for modelling

Now that I have gathered and explored the dataset, I can apply a few changes to it before actually creating models. First, I want to create simple baskets for the Local stock rate, the net sales and the quantity sold. Considering the correlation matrix, I chose to divide the local stock rate between Below_average and Above_average, splitting the data with the mean value. For the values of Net_sales and Qty_sold, on the other hand, I want to have a more granular vision and I decided to divide them in incremental quantiles in the hope to capture more information.

In [16]:
```
# Obtaining all necessary statistical values from the describe function
final_data_JP.describe()
```

Out[16]:

	Local_stock_rate	Net_sales	Qty_sold	Enrichment
count	[redacted]	[redacted]	[redacted]	1186.000000
mean	[redacted]	[redacted]	[redacted]	0.666948
std	[redacted]	[redacted]	[redacted]	0.471504
min	[redacted]	[redacted]	[redacted]	0.000000
25%	[redacted]	[redacted]	[redacted]	0.000000
50%	[redacted]	[redacted]	[redacted]	1.000000
75%	[redacted]	[redacted]	[redacted]	1.000000
max	[redacted]	[redacted]	[redacted]	1.000000

In [17]:
```
# Creating 2 baskets for the local stock rate column with a simple lambda func
tion
final_data_JP['Local_stock_rate'] = final_data_JP['Local_stock_rate'].apply(
    lambda x: 'Above_average' if x > [redacted] else 'Below_average')


# Creating 4 baskets for the net sales using a function
def split_net_sales(row):
    if row['Net_sales'] >= [redacted]:
        val = '>75%'
    elif (row['Net_sales'] >= [redacted] and row['Net_sales'] < [redacted]):
        val = '50-75%'
    elif (row['Net_sales'] >= [redacted] and row['Net_sales'] < [redacted]):
        val = '25-50%'
    else:
        val = '<25%'
    return val

final_data_JP['Net_sales'] = final_data_JP.apply(split_net_sales, axis=1)


# Creating 4 baskets for the quantity sold using a function
def split_qty_sold(row):
    if row['Qty_sold'] >= [redacted]:
        val = '>75%'
    elif (row['Qty_sold'] >= [redacted] and row['Qty_sold'] < [redacted]):
        val = '50-75%'
    elif (row['Qty_sold'] >= [redacted] and row['Qty_sold'] < [redacted]):
        val = '25-50%'
    else:
        val = '<25%'
    return val

final_data_JP['Qty_sold'] = final_data_JP.apply(split_qty_sold, axis=1)
```

In [18]:
```
# Checking the results
final_data_JP.head()
```

Out[18]:

	PSF	Technology	Local_stock_rate	Net_sales	Qty_sold	Enrichment
0	PSF_432042	TCFM	Above_average	>75%	>75%	1
1	PSF_437181	A&C	Above_average	>75%	50-75%	1
2	PSF_437185	A&C	Below_average	>75%	<25%	1
3	PSF_409441	TCFM	Above_average	50-75%	>75%	1
4	PSF_409464	TCFM	Below_average	>75%	>75%	1

One last transformation I need is about the `Enrichment` column. I will transform the 2 labels of `Enriched` / `Not enriched` into `1`'s and `0`'s respectively.

Now that all data are appropriately assigned to their respective baskets, I can split the dataset to apply the models.

In [19]:
```
# First I create the dataset with the features. To do this, I will use one-hot encoding
X = pd.get_dummies(final_data_JP.drop(['PSF', 'Enrichment'], axis=1))

# Then I create the vector for the classes
y = final_data_JP['Enrichment']

# Finally I split the dataset using the train_test_split function
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 81)
```

Now I have everything I need to apply different models and compare results.

4. Applying models

I will now deploy 3 models on the dataset split in train and test above, and I will evaluate all of them with the same standards in order to chose the one that gives me the best results. Once I establish the best performing option, I will apply it to a dataset that doesn't have any labeled values about enrichment.

Obviously I will apply the same evaluations to all 3 models, therefore I will create a function that stores already all the commands that I need. This allows me to apply the same function iteratively without having to re-write the same code.

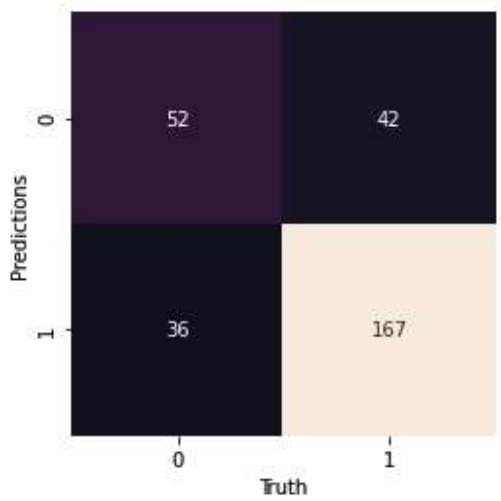
In [20]:
```
# Creating a function to evaluate all models
def model_and_train_evaluation(model, X_train, y_train, X_test, y_test):

    # Fitting the model
    model.fit(X_train, y_train)

    # Creating predictions
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    # Calculating and printing accuracy and recall scores
    print("Train accuracy: ", metrics.accuracy_score(y_train, y_pred_train))
    print('Train recall: ', metrics.recall_score(y_train, y_pred_train, pos_la
bel = 1))
    print("Test accuracy: ", metrics.accuracy_score(y_test, y_pred_test))
    print('Test recall: ', metrics.recall_score(y_test, y_pred_test, pos_label
= 1))

    # Visualization of the confusion matrix for the model
    y_true = y_test.tolist()
    mat = confusion_matrix(y_true, y_pred_test)
    sns.heatmap(mat.T, square = True, annot = True, fmt = 'd', cbar= False)
    plt.xlabel('Truth')
    plt.ylabel('Predictions')
```

4.1. Random forest

The first model I will apply is the random forest.

In [21]:
```
# Creating a model for random forest
random_forest = RandomForestClassifier()

# Calling the evaluation function on this model
model_and_train_evaluation(random_forest, X_train, y_train, X_test, y_test)
```

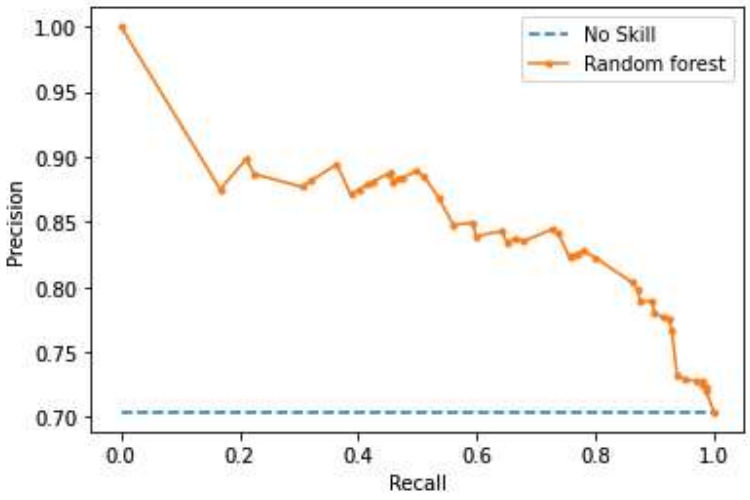
```
Train accuracy:  0.7446569178852643
Train recall:  0.8127147766323024
Test accuracy:  0.7373737373737373
Test recall:  0.7990430622009569
```



In the following diagram I am going to calculate the precision (proportion of correct predictions captured by the model) and the recall (proportion of correct predictions missed in the model) with a **precision-recall curve**.

In [22]:
```
# Calculating the predictions probabilities based on the test subset
random_forest_probs = random_forest.predict_proba(X_test)

# Isoltaing the probabilities for the positive outcome only
random_forest_probs = random_forest_probs[:, 1]

# Calculating precision and recall, with respective curves
yhat = random_forest.predict(X_test)
rf_precision, rf_recall, _ = precision_recall_curve(y_test, random_forest_prob
s)
rf_f1, rf_auc = f1_score(y_test, yhat), auc(rf_recall, rf_precision)

# Calculating the scores for F1 and area under the curve
print('Random forest: f1=%.3f auc=%.3f' % (rf_f1, rf_auc))

# Plotting the precision-recall curves
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(rf_recall, rf_precision, marker='.', label='Random forest')

# Setting up and visualizing the plot
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()
pyplot.show()
```

```
Random forest: f1=0.811 auc=0.859
```



As a rule of thumb, the more the curve is moving towards the upper right corner, the more solid are our results. This model seems to have quite an average performance, but we can use the values of **F1** and the **Area under the curve** or **AUC** to grasp this mathematically.

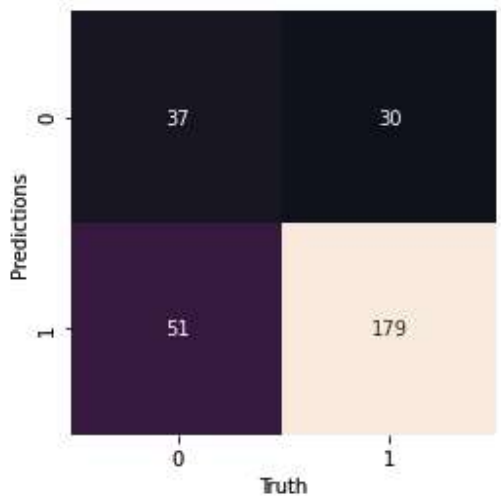
The F1 score (i.e. the weighted average or precision and recall) is equal to 81%. The **AUC** (i.e. the measure of separability between precision and recall) is about 86%. Both these results are good but not impressive.

4.2. Logistic regression

The second model I will apply is logistic regression.

In [23]:
```
# Creating a model for logistic regression
logistic_regression = LogisticRegression(solver='newton-cg')

# Calling the evaluation function on this model
model_and_train_evaluation(logistic_regression, X_train, y_train, X_test, y_te
st)
```

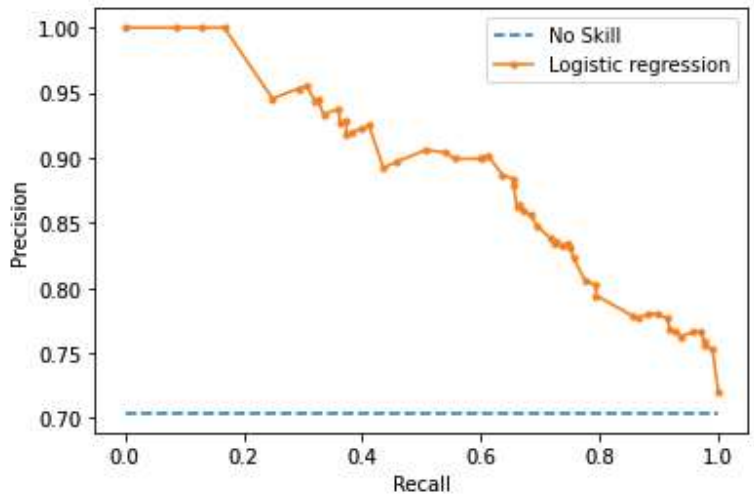
```
Train accuracy:  0.7289088863892014
Train recall:  0.8539518900343642
Test accuracy:  0.7272727272727273
Test recall:  0.8564593301435407
```



And just as before, I will also visualize the precision-recall curve.

In [24]:
```
# Calculating the predictions probabilities based on the test subset
logistic_regression_probs = logistic_regression.predict_proba(X_test)

# Isoltaing the probabilities for the positive outcome only
logistic_regression_probs = logistic_regression_probs[:, 1]

# Calculating precision and recall, with respective curves
yhat = logistic_regression.predict(X_test)
lr_precision, lr_recall, _ = precision_recall_curve(y_test, logistic_regressio
n_probs)
lr_f1, lr_auc = f1_score(y_test, yhat), auc(lr_recall, lr_precision)

# Calculating the scores for F1 and area under the curve
print('Logistic regression: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))

# Plotting the precision-recall curves
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logistic regression')

# Setting up and visualizing the plot
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()
pyplot.show()
```

```
Logistic regression: f1=0.815 auc=0.894
```



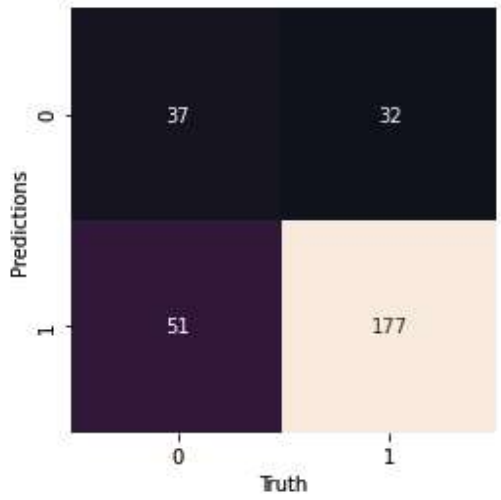
Evaluating the information contained in this graph, we can see that the curve pushes far more on the upper right than the previous model. Also **F1** is equal to 81.5% and **AUC** to 89,4%, both higher than the random forest's results.

4.3. Support vector machine

The third model I will apply is support vector machine.

In [25]:
```
# Creating a model for support vector machine
svm = SVC(kernel='linear', C=0.025, random_state=81, probability=True)

# Calling the evaluation function on this model
model_and_train_evaluation(svm, X_train, y_train, X_test, y_test)
```

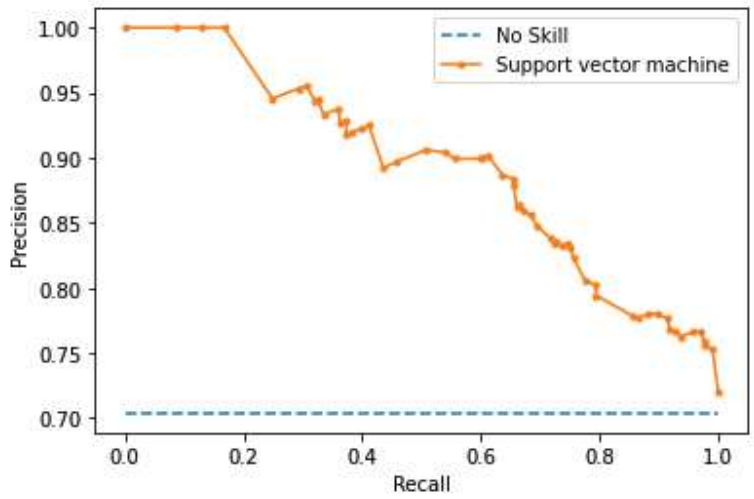
```
Train accuracy:  0.7221597300337458
Train recall:  0.8281786941580757
Test accuracy:  0.7205387205387206
Test recall:  0.84688995215311
```



And now for the precision-recall curve.

In [26]:
```
# Calculating the predictions probabilities based on the test subset
svm_probs = svm.predict_proba(X_test)

# Isoltaing the probabilities for the positive outcome only
svm_probs = svm_probs[:, 1]

# Calculating precision and recall, with respective curves
yhat = svm.predict(X_test)
svm_precision, svm_recall, _ = precision_recall_curve(y_test, svm_probs)
svm_f1, svm_auc = f1_score(y_test, yhat), auc(svm_recall, svm_precision)

# Calculating the scores for F1 and area under the curve
print('Support vector machine: f1=%.3f auc=%.3f' % (svm_f1, svm_auc))

# Plotting the precision-recall curves
no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Support vector machin
e')

# Setting up and visualizing the plot
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()
pyplot.show()
```

```
Support vector machine: f1=0.810 auc=0.854
```



For our final evaluation, we can see that the curve is still quite nicely pointing towards the upper right of the diagram. **F1** (at 81%) and **AUC** (at 85.4%) are better than our first model but lower than the logistic regression.

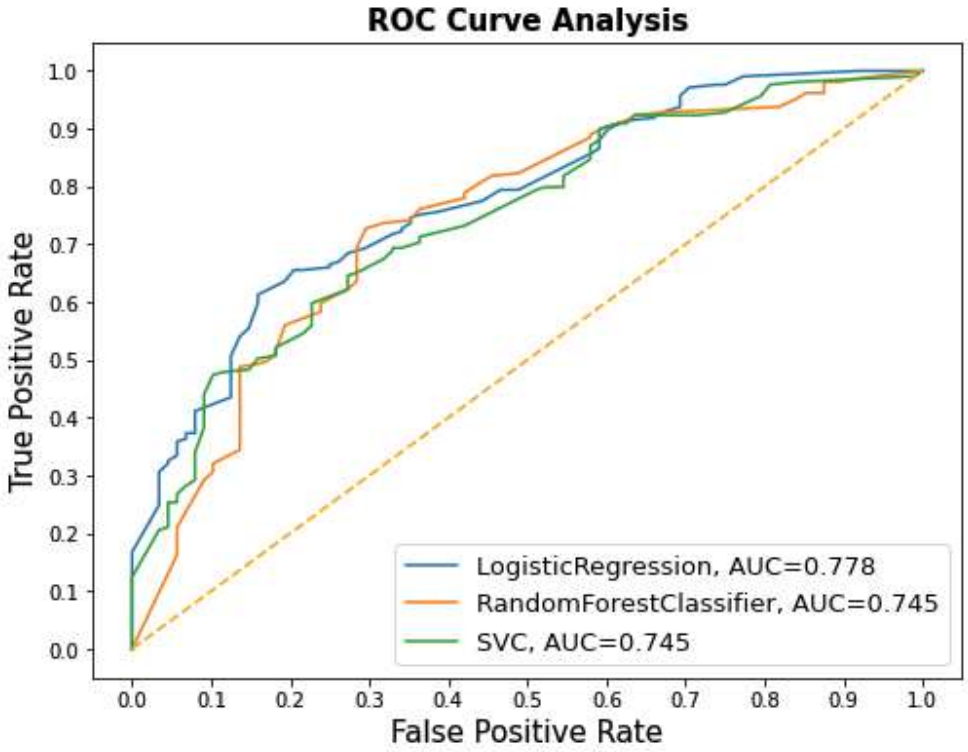
4.4. Final evaluation

Logistic regression seems to perform slightly better than the other models, since it gives me the highest number of true positive, other things being equal. Since we need only to decide whether to post-edit a product page, I am not really worried about the number of errors that we might commit. It is better to have a larger number of positives, since the cost of false positive and false negative wouldn't be that high for the company.

There is another evaluation that I can use, namely the **ROC curve**. Although this curve is generally used for data that are almost equally distributed between two classes, I believe that it can provide me with a last bit of information to double check my final decision on the model to pick.

In [27]:
```
# Creating a list of classifiers with all the models I used
classifiers = [LogisticRegression(random_state=81),
               RandomForestClassifier(random_state=81),
              SVC(random_state=81, probability=True)]

# Creating a dataframe that sums up the classifiers, the false positive rate,
# the true positive rate and the area under the curve
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

# Creating a loop that gathers all necessary information from the different mo
dels,
# e.g. probabilities of the predictions, ROC curve etc.
for cls in classifiers:
    model = cls.fit(X_train, y_train)
    yproba = model.predict_proba(X_test)[::,1]

    fpr, tpr, _ = roc_curve(y_test,  yproba)
    auc = roc_auc_score(y_test, yproba)

    result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

# Setting name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

# Creating the plot
fig = plt.figure(figsize=(8,6))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

# Setting details like style of the lines, labels and ticks
plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

# Giving a title and a legend to the plot and showing the plot
plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')
```

Out[27]: <matplotlib.legend.Legend at 0x1c284733a58>



With the amount and quality of data that I have at my disposal, it seems that the **logistic regression** is doing a slightly better job at gaining information than the other two models in all evaluations.

Given the limited scope of the project, I will not further fine tune the models at this stage but I will limit my analysis to what I have been discovering so far.

5. Applying the model to a second dataset

To give my answer to the initial brief, I will deploy the algorithm used for the logistic regression classifier to predict outcomes on a completely new dataset, that doesn't have a truth already set by observation.

First of all, I will import a similar dataset based on the **Chinese market** results. It is important to note that I was informed by my Line Manager that the two markets are similar when it comes to enrichment needs, as noted in the paragraph ***Assumptions***.

The preparation of this second dataset will mirror the preparation used for `final_data_JP` in all its points. I will directly import the final form of this second file here, to avoid repeating steps already detailed above.

In [28]:
```
# Importing the Chinese dataset
final_data_CH = pd.read_excel('data_CH.xlsx')

# Visualizing the dataset to check for correct implementation
final_data_CH.head()
```

Out[28]:

	PSF	Technology	Local_stock_rate	Net_sales	Qty_sold
0	PSF_432042	TCFM	Below_average	>75%	>75%
1	PSF_409441	TCFM	Below_average	50-75%	50-75%
2	PSF_412530	A&C	Below_average	50-75%	25-50%
3	PSF_412607	A&C	Below_average	<25%	<25%
4	PSF_412623	A&C	Below_average	>75%	50-75%

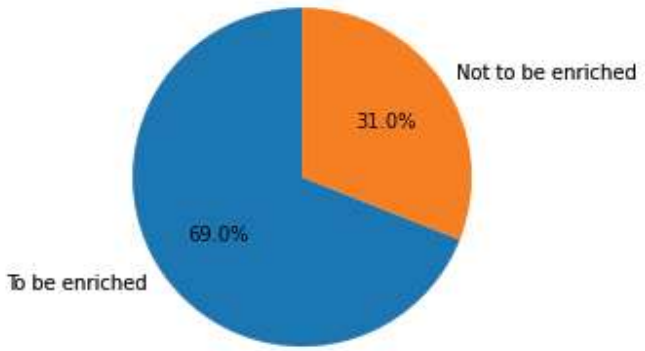
In [29]:
```
# Creating a one-hot encoded version of the dataset
X_CH = pd.get_dummies(final_data_CH.drop(['PSF'], axis=1))

# Storing the predictions for the dataset in a new column
# The predictions are based on the logistic regression model used above
final_data_CH['Enrichment'] = logistic_regression.predict(X_CH)

# Visualizing the final dataset
final_data_CH.head()
```

Out[29]:

	PSF	Technology	Local_stock_rate	Net_sales	Qty_sold	Enrichment
0	PSF_432042	TCFM	Below_average	>75%	>75%	1
1	PSF_409441	TCFM	Below_average	50-75%	50-75%	0
2	PSF_412530	A&C	Below_average	50-75%	25-50%	1
3	PSF_412607	A&C	Below_average	<25%	<25%	1
4	PSF_412623	A&C	Below_average	>75%	50-75%	1

The calculation of metrics for this model is of course meaningless, since I couldn't really train the model on a list of true values observed in the real world. However, I can visualize the number of families to be enriched and compare with the previous dataset.

In [30]:
```
# To obtain a pie chart, first I calculate how many families of products in to
tal have been enriched or not enriched
slices_CH = [final_data_CH[final_data_CH['Enrichment'] == 1].count().PSF,
          final_data_CH[final_data_CH['Enrichment'] == 0].count().PSF]

# Then I select the labels I want for the two sections of the pie
enrichment = ['To be enriched', 'Not to be enriched']

# Selecting the correct function in matplotlib and launch the visualization
plt.pie(slices_CH, labels=enrichment, startangle=90, autopct='%.1f%%')
plt.show()

print('Number of families to enrich: ', final_data_CH[final_data_CH['Enrichmen
t'] == 1].count().PSF)
```




```
Number of families to enrich:  854
```

These results are very similar to the previous dataset - where we had a 66.7% of families enriched for the Japanese market, which makes me pretty confident that my predictions are not completely out of line.

Lastly, I create an excel file with the list of families to be enriched, to share with my colleagues in China.

In [31]:
```
# Creating the list of families to be enriched
list_for_china = final_data_CH[final_data_CH['Enrichment'] == 1]

# Saving the list of families in an Excel file
list_for_china.to_excel("Enrichment_list(China).xlsx")
```

6. Future actions

The list of families created by deploying the model can now be handed to my colleagues of the Chinese market. These data will inform their internal decision on how to tackle the enrichment task.

This result is clearly actionable (there is a defined list to prioritize, even if probably more data will give a deeper insight) and measurable (the team will have a precise number of families to be worked upon and can therefore plan in advance).

Further exploration of larger datasets and cross validation among countries could certainly lead to more insight. I have presented this idea to my Line Manager, who will be deciding about future use of this analysis.