Reference

Data Camp course.

Building Logistic Regression Models

Exploring the base table

In [1]:
```
import pandas as pd
basetable = pd.read_csv('basetable_ex2_4.csv')
# print(basetable.head())
#The above table is different from the table in DataCamp

population_size  = len(basetable)
print(population_size)
targets_count = sum(basetable["target"]) #target has value of 0 and 1.
print(targets_count)
print(targets_count/population_size)
```

```
25000
1187
0.04748
```

In [2]:
```
basetable.head()
```

Out[2]:

	target	gender_F	income_high	income_low	country_USA	country_India	country_UK	age	time_since_last_gift	time_since_first_gift	r
0	0	1	0	1	0	1	0	65	530	2265	
1	0	1	0	0	0	1	0	71	715	715	
2	0	1	0	0	0	1	0	28	150	1806	
3	0	1	0	1	1	0	0	52	725	2274	
4	0	1	1	0	1	0	0	82	805	805	

Interpretation of coefficients

Built a logistic regression model to predict which donors are most likely to donate for a project, using age and time_since_last_gift (number of months since the last gift) as predictors. The output of the logistic regression model is as follows:

y = 0.3 + 4.5 * age - 2.3 * time_since_last_gift Which of the following statements holds, according to the model?

Answer: Older donors that recently donated are most likely to donate.
How the coefficients for logistic regression are calculated? Should be related to the term $\beta^T x$ in the sigmoid function. Normally the output of the probabilities $p(y \mid x)$ in the logistic regression is used.

Building a logistic regression model

In [2]:
```
from sklearn import linear_model

X = basetable[["age","gender_F","time_since_last_gift"]]

y = basetable["target"]

logreg = linear_model.LogisticRegression(solver = 'lbfgs')
logreg.fit(X, y)
```

Out[2]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

Showing the coefficients and intercept

Given a fitted logistic regression model logreg, we can retrieve the coefficients using the attribute `coef_` . The order in which the coefficients appear, is the same as the order in which the variables were fed to the model. The intercept can be retrieved using the attribute `intercept_` .

In [33]:
```
predictors = ["age","gender_F","time_since_last_gift"]
X = basetable[predictors].values
y = basetable["target"].values
logreg = linear_model.LogisticRegression(solver = 'liblinear')
logreg.fit(X, y)

coef = logreg.coef_
for p,c in zip(predictors,list(coef[0])):
    print(p + '\t' + str(c))

# Assign the intercept to the variable intercept
intercept = logreg.intercept_
print(intercept)
```

```
age     0.007178355659086629
gender_F        0.11430414536348246
time_since_last_gift    -0.00130875011331457
[-2.54149728]
```

Donor that is most likely to donate

predictions below should be a DataFrame but not a numpy array obtained earlier. So the code below cannot be run.

In [ ]:
```
predictions_sorted = predictions.sort(["probability"])
print(predictions_sorted.tail(1)) #This gives the most probable person to donate.
```

Forward stepwise variable selection for logistic regression

Which model is best?

For 4 models:
A: A model with 10 variables that has an AUC of 0.76
B: A model with 10 variables that has an AUC of 0.73
C: A model with 15 variables that has an AUC of 0.76
D: A model with 15 variables that has an AUC of 0.73.
Which model is best, assuming all variables are equally easy to calculate and maintain.

Answer: A.

Calculating AUC

The AUC value assesses how well a model can order observations from low probability to be target to high probability to be target. In Python, the roc_auc_score function can be used to calculate the AUC of the model. It takes the true values of the target and the predictions as arguments.

In [35]:
```
from sklearn.metrics import roc_auc_score
predictions = logreg.predict_proba(X)
predictions_target = predictions[:,1] # Second column contains the predictions fr the target.

auc = roc_auc_score(y, predictions_target)
print(round(auc,2))
```

```
0.63
```

Verify the following comments in the future. In logistic regerssion, predictions_target above is like providing a hypothesis function $h_\theta(x)$. By setting different threshold T, once can plot the ROC curve with the help of true values of y.

Using different sets of variables

Adding more variables and therefore more complexity to your logistic regression model does not automatically result in more accurate models.

In [36]:
```
variables_1 = ['mean_gift', 'income_low']
variables_2 = ['mean_gift', 'income_low', 'gender_F', 'country_India', 'age']

X_1 = basetable[variables_1]
X_2 = basetable[variables_2]
y = basetable["target"]

logreg = linear_model.LogisticRegression(solver = 'liblinear')

logreg.fit(X_1, y)
predictions_1 = logreg.predict_proba(X_1)[:,1]
auc_1 = roc_auc_score(y, predictions_1)

logreg.fit(X_2, y)
predictions_2 = logreg.predict_proba(X_2)[:,1]
auc_2 = roc_auc_score(y, predictions_2)

print(round(auc_1,2))
print(round(auc_2,2))
```

```
0.68
0.69
```

You can see that the model with 5 variables has the same AUC as the model using only 2 variables. Adding more variables doesn't always increase the AUC. In DataCamp, the output results are same. Both are 0.69.

Selecting the next best variable

The forward stepwise variable selection method starts with an empty variable set and proceeds in steps, where in each step the next best variable is added. To implement this procedure, two handy functions have been implemented as below.

In [37]:
```
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
def auc(variables, target, basetable):
    X = basetable[variables].values
    y = basetable[target].values
    logreg = linear_model.LogisticRegression(solver = 'liblinear')
    logreg.fit(X, y.ravel())
    #y.ravel() change column vector (25000,1) to 1d array (25000)
    #Otherwise logreg.fit() will complain

    predictions = logreg.predict_proba(X)[:,1]
    auc_value = roc_auc_score(y.ravel(), predictions)
    return(auc_value)
auc1 = auc(["age","gender_F"],["target"],basetable)
print(round(auc1,2))
```

```
0.54
```

In [38]:
```
def next_best(current_variables,candidate_variables, target, basetable):
    best_auc = -1
    best_variable = None
    for v in candidate_variables:
        auc_v = auc(current_variables + [v], target, basetable)
        if auc_v >= best_auc:
            best_auc = auc_v
            best_variable = v
    return best_variable

current_variables = ["age","gender_F"]
candidate_variables = ["min_gift","max_gift","mean_gift"]
next_variable = next_best(current_variables, candidate_variables,["target"], basetable)
print(next_variable)
# min_gift
```

```
max_gift
```

In [39]:
```
auc_current = auc(['max_gift', 'mean_gift', 'min_gift'], ["target"], basetable)
print(round(auc_current,4))

next_variable = next_best(['max_gift', 'mean_gift', 'min_gift'], ['age', 'gender_F'], ["target"],basetable)
print(next_variable)

auc_current_age = auc(['max_gift', 'mean_gift', 'min_gift', 'age'], ["target"], basetable)
print(round(auc_current_age,4))

auc_current_gender_F = auc(['max_gift', 'mean_gift', 'min_gift','gender_F'], ["target"], basetable)
print(round(auc_current_gender_F,4))
```

```
0.7125
age
0.7148
0.713
```

The model that has age as next variable has a better AUC than the model that has gender_F as next variable. Therefore, age is selected as the next best variable.

Finding the order of variables

The forward stepwise variable selection procedure starts with an empty set of variables, and adds predictors one by one. In each step, the predictor that has the highest AUC in combination with the current variables is selected.

In [47]:
```
candidate_variables = list(basetable.columns.values)
candidate_variables.remove("target")

current_variables = []

# The forward stepwise variable selection procedure
number_iterations = 100
for i in range(0, number_iterations):
    next_variable = next_best(current_variables, candidate_variables, ["target"], basetable)
    if(next_variable != None):
        current_variables = current_variables + [next_variable]
    if (next_variable in candidate_variables):
        candidate_variables.remove(next_variable)

    if(next_variable != None):
        print("Variable added in step " + str(i+1)  + " is " + next_variable + ".")
print(current_variables)
```

```
Variable added in step 1 is max_gift.
Variable added in step 2 is number_gift.
Variable added in step 3 is time_since_last_gift.
Variable added in step 4 is mean_gift.
Variable added in step 5 is age.
Variable added in step 6 is income_high.
Variable added in step 7 is time_since_first_gift.
Variable added in step 8 is country_USA.
Variable added in step 9 is country_India.
Variable added in step 10 is min_gift.
Variable added in step 11 is country_UK.
Variable added in step 12 is gender_F.
Variable added in step 13 is income_low.
['max_gift', 'number_gift', 'time_since_last_gift', 'mean_gift', 'age', 'income_high', 'time_since_first_gift',
'country_USA', 'country_India', 'min_gift', 'country_UK', 'gender_F', 'income_low']
```

Comments: Note if I use solver = 'lbfgs' in the logistic regression model, then I will have convergence problems. The following is from web:

Normally when an optimization algorithm does not converge, it is usually because the problem is not well-conditioned, perhaps due to a poor scaling of the decision variables. There are a few things you can try.

- Normalize your training data so that the problem hopefully becomes more well conditioned, which in turn can speed up convergence. One possibility is to scale your data to 0 mean, unit standard deviation using Scikit-Learn's StandardScaler for an example. Note that

you have to apply the StandardScaler fitted on the training data to the test data.

- Make sure the other arguments such as regularization weight, C, is set appropriately.
- Set max_iter to a larger value. The default is 1000.

Correlated variables

It can happen that a good variable is not added because it is highly correlated with a variable that is already in the model. You can test this calculating the correlation between these variables.

In [48]:
```
import numpy as np
auc_min_gift = auc(["min_gift"],["target"],basetable)
print(round(auc_min_gift,2))

auc_income_high = auc(["income_high"],["target"],basetable)
print(round(auc_income_high,2))

correlation = np.corrcoef(basetable["min_gift"], basetable["mean_gift"])[0,1]
print(round(correlation,2))
```

```
0.57
0.52
0.76
```

Partitioning into train and test sets

- This division between train and test sets is done randomly, but when the target incidence is low, it could be necessary to stratify, that is, to make sure that the train and test data contain an equal percentage of targets.
- Partition the data with stratification and verify that the train and test data have equal target incidence.

In [14]:
```
from sklearn.model_selection import train_test_split

X = basetable.drop('target', 1)
y = basetable["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.50, stratify = y)

train = pd.concat([X_train, y_train], axis=1)
test = pd.concat([X_test, y_test], axis=1)

print(round(sum(train["target"])/len(train), 2))
print(round(sum(test["target"])/len(test), 2))
```

```
0.05
0.05
```

Evaluating a model on test and train

In [62]:
```
from sklearn import linear_model
from sklearn.metrics import roc_auc_score
def auc_train_test(variables, target, train, test):
    X_train = train[variables]
    X_test = test[variables]
    Y_train = train[target].values
    Y_test = test[target].values
    logreg = linear_model.LogisticRegression(solver = 'liblinear')

    logreg.fit(X_train, Y_train.ravel())

    predictions_train = logreg.predict_proba(X_train)[:,1]
    predictions_test = logreg.predict_proba(X_test)[:,1]

    auc_train = roc_auc_score(Y_train.ravel(), predictions_train)
    auc_test = roc_auc_score(Y_test.ravel(),predictions_test)
    return(auc_train, auc_test)
```

In [63]:
```
from sklearn.model_selection import train_test_split

X = basetable.drop('target', 1)
y = basetable["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, stratify = y)

# Create the final train and test basetables
train = pd.concat([X_train, y_train], axis=1)
test = pd.concat([X_test, y_test], axis=1)

# Apply the auc_train_test function
auc_train, auc_test = auc_train_test(["age","gender_F"], ["target"], train, test)
print(round(auc_train,2))
print(round(auc_test,2))
```

```
0.55
0.54
```

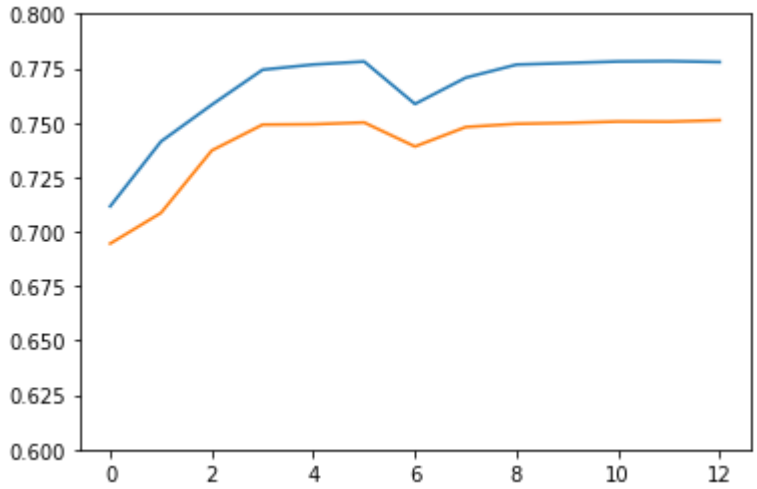
Building the AUC curves

The forward stepwise variable selection procedure provides **an order** in which variables are optimally added to the predictor set. In order to decide where to cut off the variables, you can make the train and test AUC curves. These curves plot the train and test AUC using the first, first two, first three, ... variables in the model.

The variables ordered according to the forward stepwise procedure are given in the list variables (not available here).

In [69]:

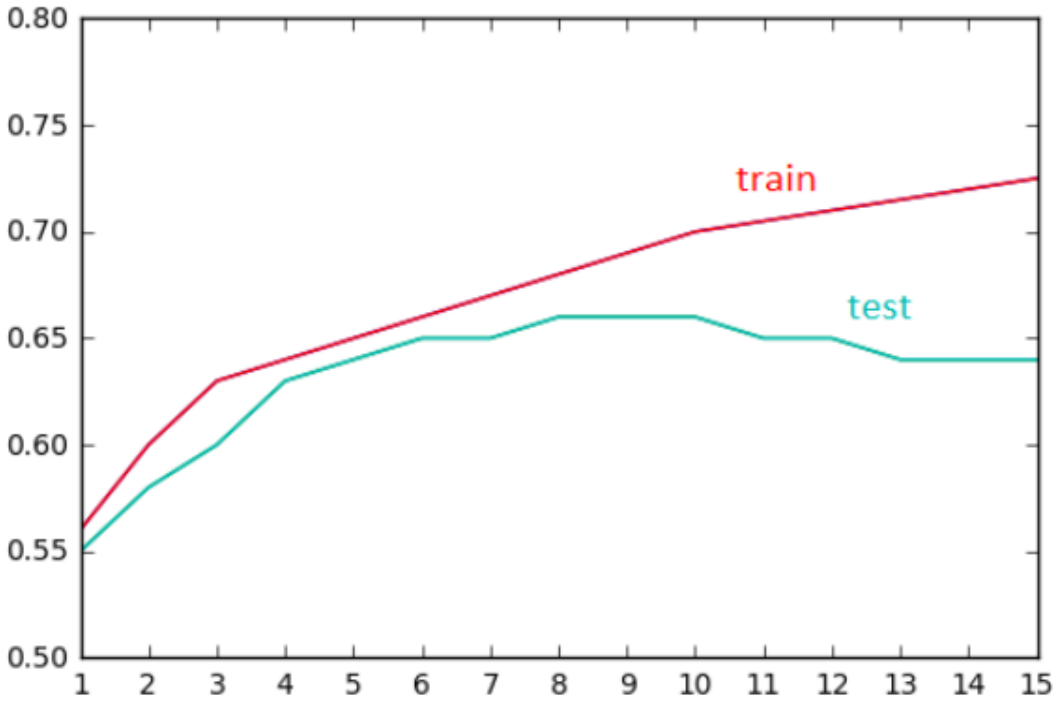
```
auc_values_train = []
auc_values_test = []

variables_evaluate = []

#The line below is newly added for code running. However, their values are different from DataCamp.
variables = ['max_gift', 'number_gift', 'time_since_last_gift', 'mean_gift', 'age', 'income_high', 'time_since_f
             'country_USA', 'country_India', 'min_gift', 'country_UK', 'gender_F', 'income_low']
#The line above is newly added

for v in variables:

    variables_evaluate.append(v)

    auc_train, auc_test = auc_train_test(variables_evaluate, ["target"], train, test)

    auc_values_train.append(auc_train)
    auc_values_test.append(auc_test)

import matplotlib.pyplot as plt
import numpy as np

x = np.array(range(0,len(auc_values_train)))
y_train = np.array(auc_values_train)
y_test = np.array(auc_values_test)
# plt.xticks(x, variables, rotation = 90)
plt.plot(x,y_train)
plt.plot(x,y_test)
plt.ylim((0.6, 0.8))
plt.show()
```



The above AUC curves are different from that of DataCamp due to the difference in data. Here, it seems 5 variables gives best results. More variables will not help.

Deciding the cut-off

The forward stepwise variable selection results in the following AUC values. How many variables should be included in the model?



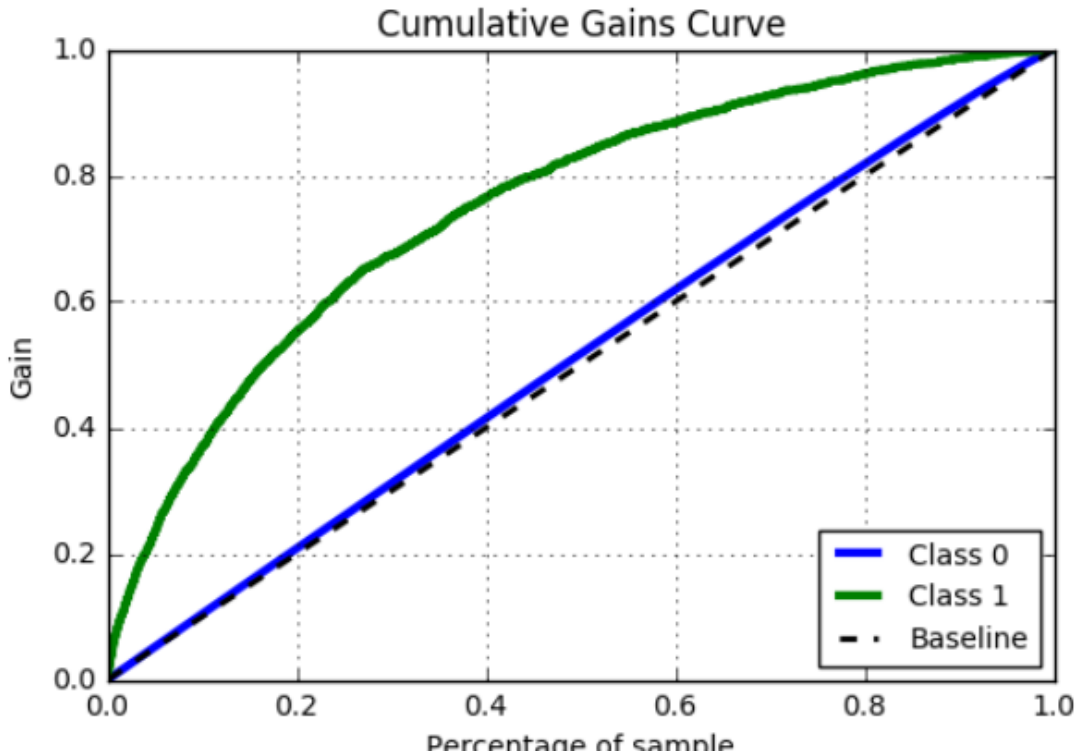
Answer: 8 variables, the test AUC does not increase after this point and it is better to have less complex models.

Explaining model performance to business

- Cumulative gains curve.
- Lift graph. See reference later.

Interpreting the cumulative gains curve

You built a model to predict which donors are most likely to react on a campaign and built a cumulative gains curve plotted below. Assume you have budget to send a letter to the top 30 000 donors among the 100 000 donors. How many targets (donors that react) will you have reached, if there are 5 000 targets among the 100 000 donors? Recall that the cumulative gains curve shows for each value x on the horizontal axis the percentage targets that is reached when using the model.



You have budget to reach 30% of the donors. The cumulative gains graph shows that you reach 70% of the 5000 targets, which is 3 500 targets.

Constructing the cumulative gains curve

The cumulative gains curve is an evaluation curve that assesses the performance of your model. It shows the percentage of targets reached when considering a certain percentage of your population with the highest probability to be target according to your model.

http://mlwiki.org/index.php/Cumulative_Gain_Chart (http://mlwiki.org/index.php/Cumulative_Gain_Chart)
https://community.tibco.com/wiki/gains-vs-roc-curves-do-you-understand-difference (https://community.tibco.com/wiki/gains-vs-roc-curves-do-you-understand-difference)

Also see summary in learning theory.

In [ ]:
```
import matplotlib.pyplot as plt

import scikitplot as skplt

#Plot the cumulative gains graph
skplt.metrics.plot_cumulative_gain(targets_test, predictions_test)
plt.show()
```

In [71]:
```
!pip install scikitplot
```

```
Collecting scikitplot

  Could not find a version that satisfies the requirement scikitplot (from versions: )
No matching distribution found for scikitplot
```

A random model

In this exercise you will reconstruct the cumulative gains curve's baseline, that is, the cumulative gains curve of a random model.

To do so, you need to construct random predictions. The plot_cumulative_gain method requires two values for these predictions: one for the target to be 0 and one for the target to be 1. These values should sum to one, so a valid list of predictions could for instance be [(0.02,0.98),(0.27,0.73),...,(0.09,0.91)].

In Python, you can generate a random value between values a and b as follows:

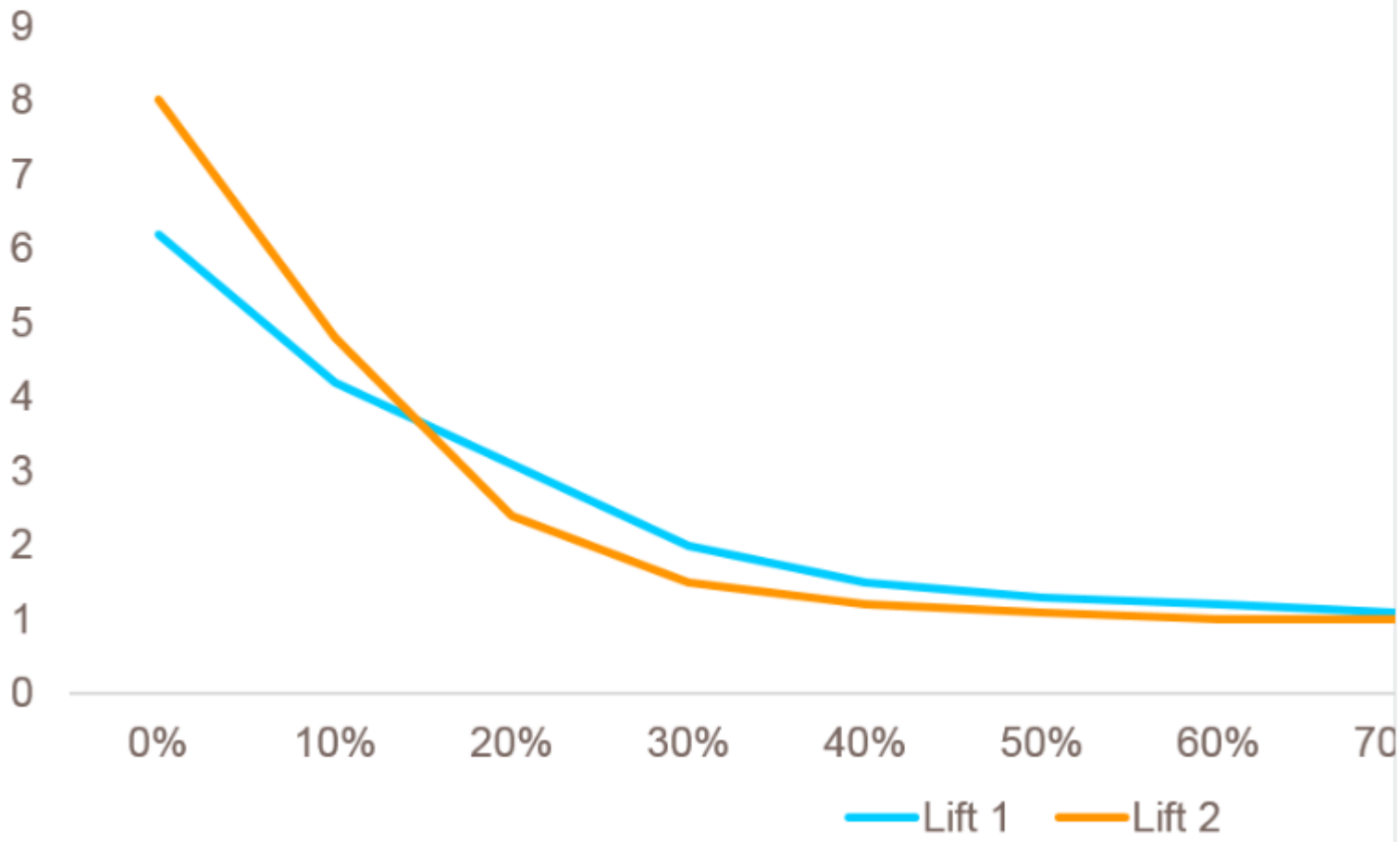
import random random_value = random.uniform(a,b) Instructions 100 XP Import the random, matplotlib and scikitplot modules Construct a list random_predictions that contains random numbers between 0 and 1. Adjust the list random_predictions such that it contains tuples (r,a) with r the original value of the list and a such that r+a=1. The true values of the target are in targets_test. Show the cumulative gains graph of your random model.

In [ ]:
```
# Import the modules
import random
import matplotlib.pyplot as plt
import scikitplot as skplt

# Generate random predictions
random_predictions = [random.uniform(0,1) for _ in range(len(targets_test))]

# Adjust random_predictions
random_predictions = [(r,1-r) for r in random_predictions]

# Plot the cumulative gains graph
skplt.metrics.plot_cumulative_gain(targets_test, random_predictions)
plt.show()
```

You can observe that the cumulative gains curve of a random model aligns with the baseline.

Interpreting the lift curve

Below are given the lift curves of two models that predict which donors are most likely to donate for a certain campaign. Assume that you have budget to address 20% of the donors. Which model should you use? Recall that the lift curve shows how many times more than average the model reaches targets when a given percentage of the population that is most likely to be target according to the model is considered.



Answer: model 1. The curve of model 1 is indeed higher at 20%, meaning that there will be more targets in the top 20% if you take model 1.

Constructing the lift curve

The lift curve is an evaluation curve that assesses the performance of your model. It shows how many times more than average the model reaches targets.

To construct this curve, you can use the plot_lift_curve method in the scikitplot module and the matplotlib.pyplot module. As for each model evaluation metric or curve, you need the true target values on the one hand and the predictions on the other hand to construct the cumulative gains curve.

Instructions 100 XP Import the matplotlib.pyplot module. Import the scikitplot module. Construct the cumulative gains curve, given that the model outputs the values in predictions_test and the true target values are in targets_test.

In [ ]:
```
# Import the matplotlib.pyplot module
import matplotlib.pyplot as plt

# Import the scikitplot module
import scikitplot as skplt

# Plot the lift curve
skplt.metrics.plot_lift_curve(targets_test, predictions_test)
plt.show()
```

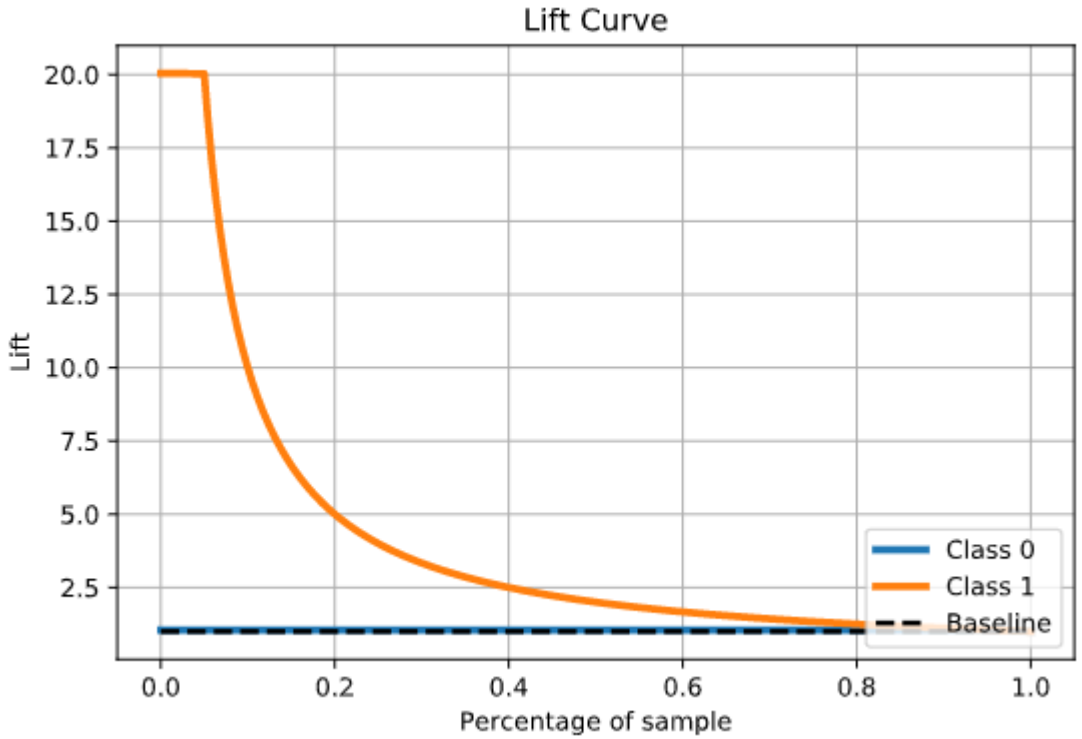
A perfect model

In this exercise you will reconstruct the lift curve of a perfect model. To do so, you need to construct perfect predictions.

Recall that the plot_lift_curve method requires two values for the predictions argument: the first argument for the target to be 0 and the second one for the target to be 1.

Instructions 100 XP Construct a list that has perfect predictions. The true values of the target are in targets_test. Plot the lift curve using the perfect predictions.

In [ ]:
```
# Generate perfect predictions
perfect_predictions = [(1-target,target) for target in targets_test["target"]]

# Plot the lift curve
skplt.metrics.plot_lift_curve(targets_test, perfect_predictions)
plt.show()
```

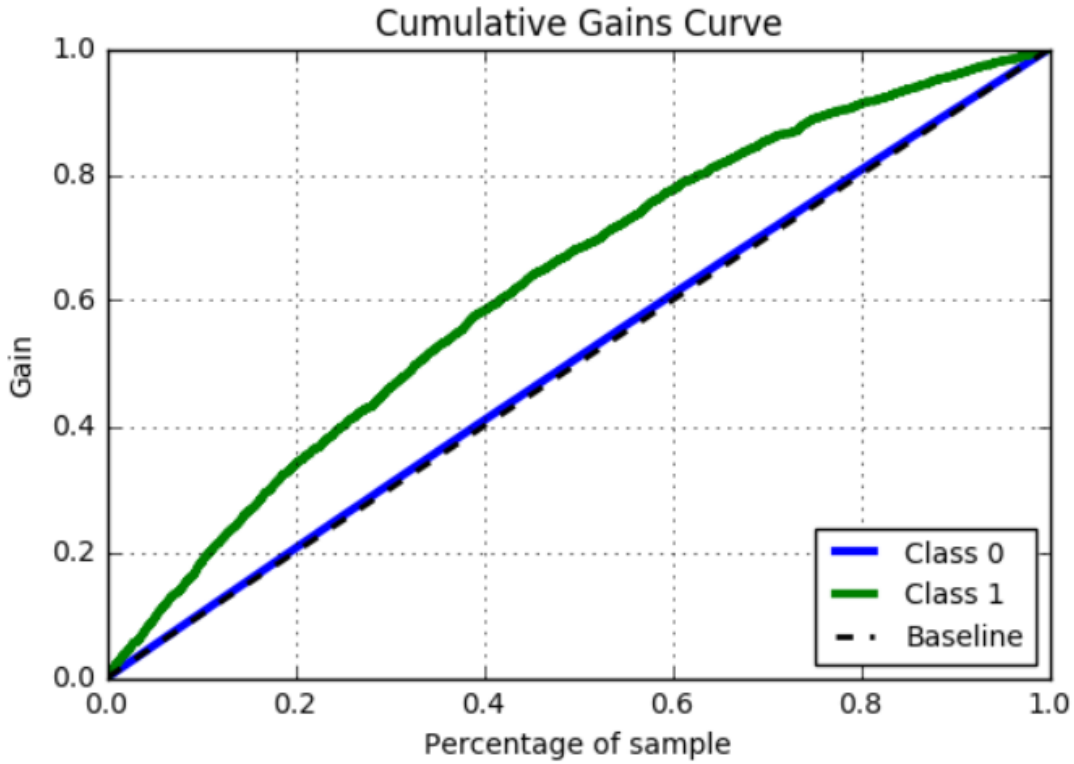


You can observe that the lift is first 20, which is normal as there are 5% targets: you can only have 20 times more than average targets. After that the lift gradually decreases because there are no targets to add anymore.

Targeting using cumulative gains curve

Consider a population of 10 000 potential donors and 1000 targets that you want to write a letter to ask if they can donate 10 Euro. You created a model with a cumulative gains graph as given below. Sending out one letter costs 1 Euro. Assume that there is 4 000 Euro budget, so you can send letters to the top 40% donors with most potential to donate. As the targets will donate 10 Euro, reaching a target results in 10 Euro incomes.

How much profit do you expect to make?



Answer: 2000
You win 6000 Euro because you reach 600 targets, and you lose 4000 Euro because you send a campaign to 4000 donors.

Business case using lift curve

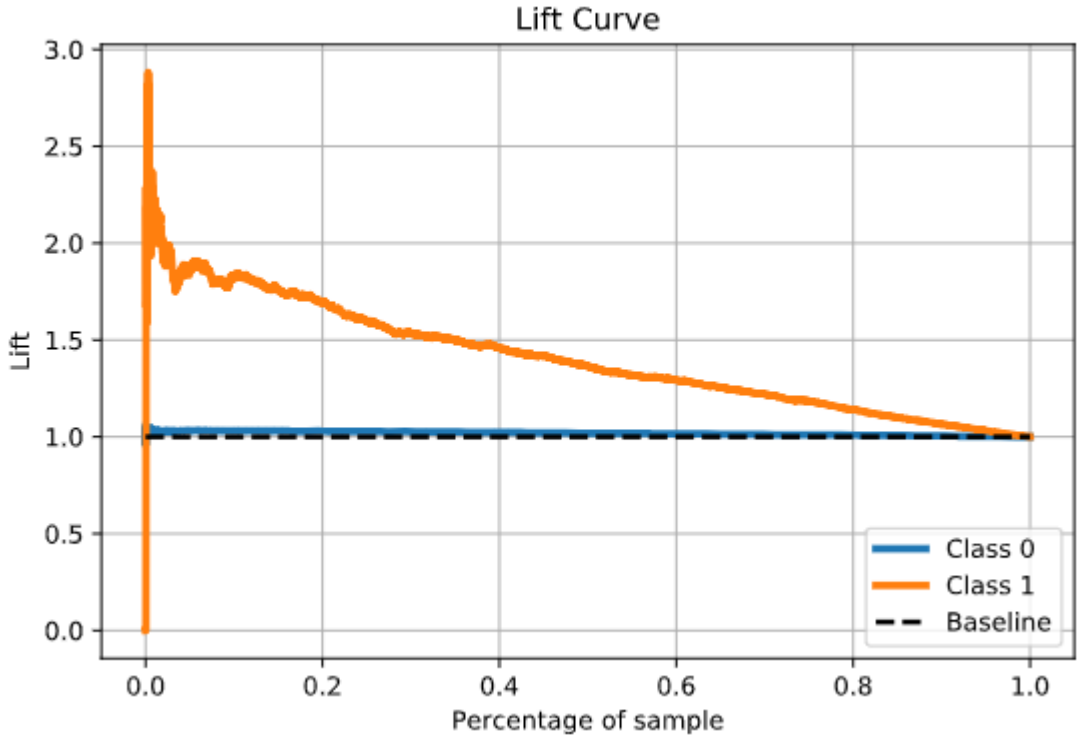
In the video you learned to implement a method that calculates the profit of a campaign:

profit = profit(perc_targets, perc_selected, population_size, campaign_cost, campaign_reward) In this method, perc_targets is the percentage of targets in the group that you select for your campaign, perc_selected the percentage of people that is selected for the campaign, population_size the total population size, campaign_cost the cost of addressing a single person for the campaign, and campaign_reward the reward of addressing a target.

In this exercise you will learn for a specific case whether it is useful to use a model, by comparing the profit that is made when addressing all donors versus the top 40% of the donors.

Instructions Plot the lift curve. The predictions are in predictions_test and the true target values are in targets_test. Read the lift value at 40% and fill it out. The information about the campaign is filled out in the script. Calculate the profit made when addressing the entire population. Calculate the profit made when addressing the top 40%.

In [ ]:
```
# Plot the lift graph
skplt.metrics.plot_lift_curve(targets_test, predictions_test)
plt.show()

# Read the lift at 40% (round it up to the upper tenth)
perc_selected = 0.4
lift = 1.5

# Information about the campaign
population_size, target_incidence, campaign_cost, campaign_reward = 100000, 0.01, 1, 100

# Profit if all donors are targeted
profit_all = profit(target_incidence, 1, population_size, campaign_cost, campaign_reward)
print(profit_all)

# Profit if top 40% of donors are targeted
profit_40 = profit(lift * target_incidence, 0.4, population_size, campaign_cost, campaign_reward)
print(profit_40)
```




```
0.0
20000.0
```

When addressing the entire donor base, you do not make any profit at all. When using the predictive model, you can make 20,000 Euro profit!

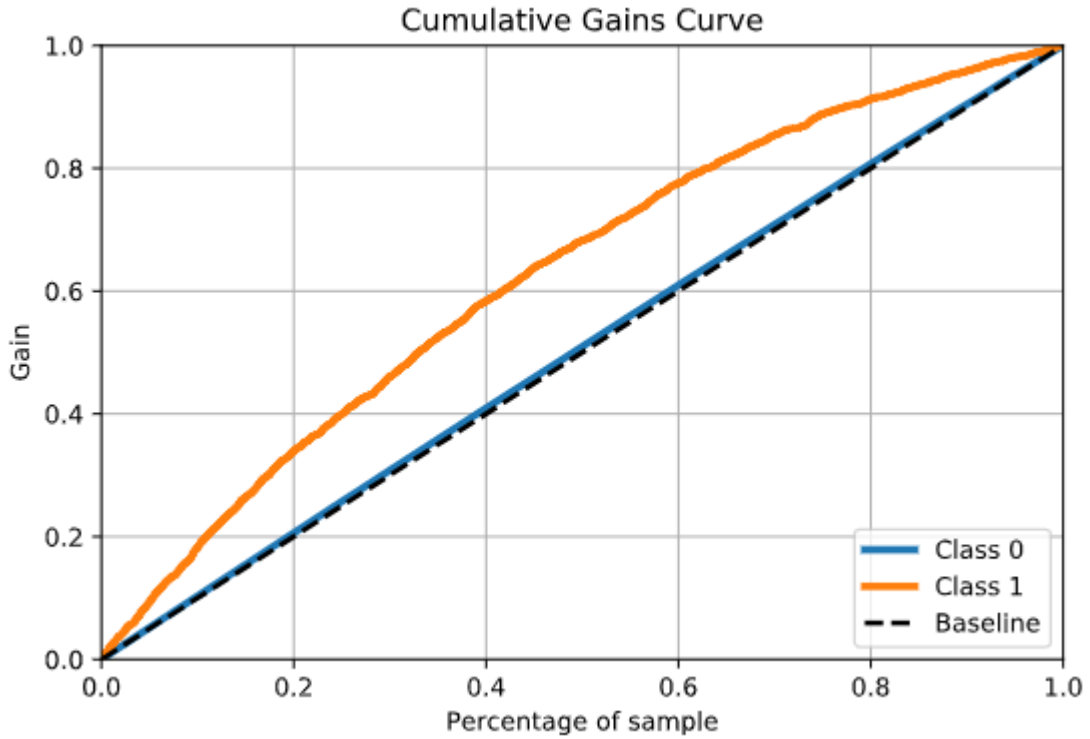
Business case using cumulative gains curve

The cumulative gains graph can be used to estimate how many donors one should address to make a certain profit. Indeed, the cumulative gains graph shows which percentage of all targets is reached when addressing a certain percentage of the population. If one knows the reward of a campaign, it follows easily how many donors should be targeted to reach a certain profit.

In this exercise, you will calculate how many donors you should address to obtain a 30 000 Euro profit.

Instructions 100 XP Plot the cumulative gains curve. The predictions are in predictions_test and the true target values are in targets_test. Assume the cost of a campaign is 0 Euro and the reward of addressing a target is 50 Euro. Fill out how many targets should be reached to make 30 000 Euro profit. There are 1000 targets in total. Fill out which percentage of the targets should be addressed. Use the cumulative gains curve to know which percentage of the population should be addressed. Round to the tenth. Given that the population consists of 10 000 donors, fill out how many donors should be addressed.

In [ ]:
```
# Plot the cumulative gains
skplt.metrics.plot_cumulative_gain(targets_test, predictions_test)
plt.show()

# Number of targets you want to reach
number_targets_toreach = 30000 / 50
perc_targets_toreach = number_targets_toreach / 1000
cumulative_gains = 0.4
number_donors_toreach = cumulative_gains * 10000
```



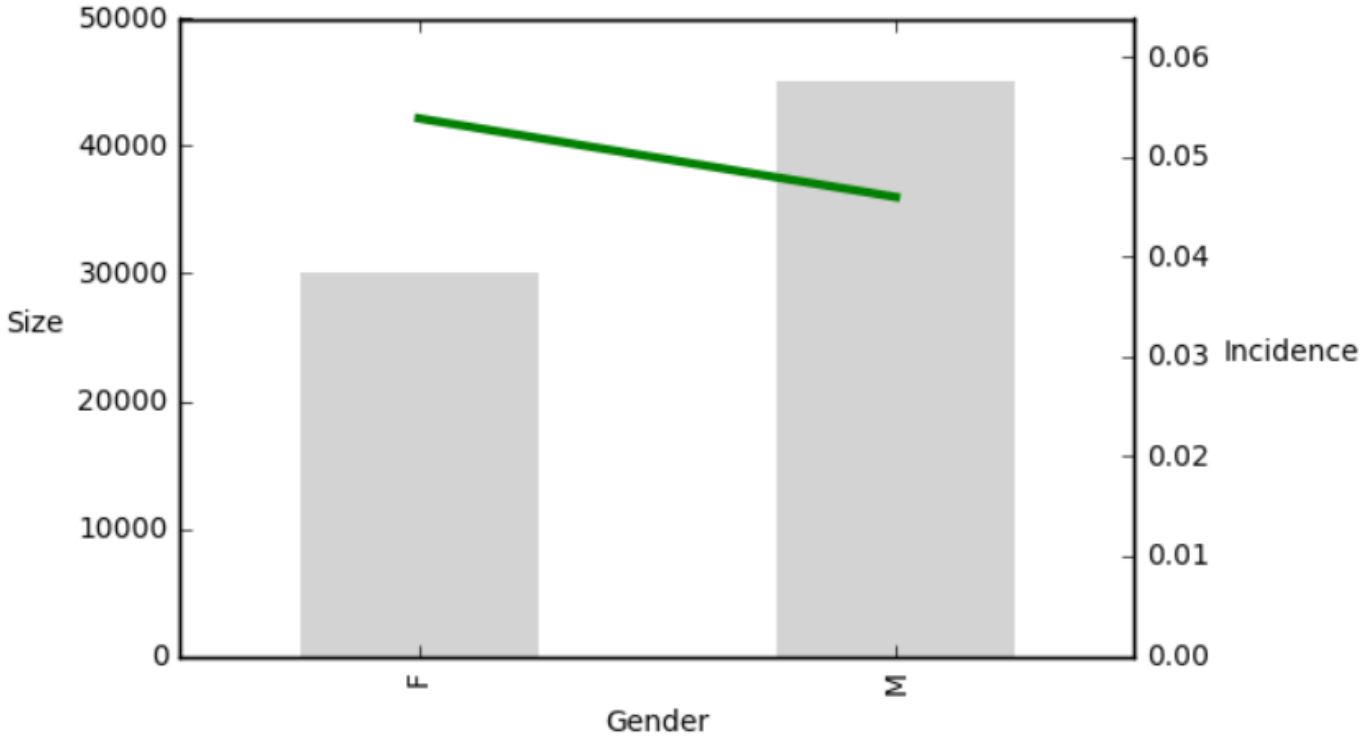
It looks like you need to address 4,000 donors!

Interpreting and explaining models

In a business context, it is often important to explain the intuition behind the model you built. Indeed, if the model and its variables do not make sense, the model might not be used. In this chapter you'll learn how to explain the relationship between the variables in the model and the target by means of predictor insight graphs.

Interpretation of predictor insight graphs

Consider the predictor insight graph that indicates the relationship between the target incidence and the gender (`M` - male, `F` - female) of the candidate donor. The bars indicate the size of the groups, the line indicates the percentage targets.



Answer: Females are more likely to donate, but there are less females than males in the database.

Retrieving information from the predictor insight table

The predictor insight graph table contains all the information needed to construct the predictor insight graph. For each value the predictor takes, it has the number of observations with this value and the target incidence within this group. The predictor insight graph table of the predictor Country is loaded as a pandas object pig_table. You can access elements using indexing. For instance, to retrieve the target incidence of donors living in the UK, you can use:

pig_table["Incidence"][pig_table["Country"]=="UK"] Instructions 100 XP Print the number of UK donors. Print the target incidence of USA donors. Print the target incidence of India donors.

In [ ]:
```
# Inspect the predictor insight graph table of Country
print(pig_table)

# Print the number of UK donors
print(pig_table["Size"][pig_table["Country"]=="UK"])

# Check the target incidence of USA and India donors
print(pig_table["Incidence"][pig_table["Country"]=="USA"])
print(pig_table["Incidence"][pig_table["Country"]=="India"])
```

```
  Country   Size  Incidence
0   India  49849       0.05
1      UK  10057       0.05
2     USA  40094       0.05
1    10057
Name: Size, dtype: int64
2    0.05
Name: Incidence, dtype: float64
0    0.05
Name: Incidence, dtype: float64

The target incidence of USA and India donors is the same, indicating that country is not a good variable
to predict donations.
```

Discretization of a certain variable

In order to make predictor insight graphs for continuous variables, you first need to discretize them. In Python, you can discretize pandas columns using the qcut method.

To check whether the variable was nicely discretized, you can verify that the bins have equal size using the groupby method:

print(basetable.groupby("discretized_variable").size() Instructions 100 XP Use the method qcut to discretize the variable time_since_last_donation in 10 groups. Assing this variable to a new column called "bins_recency". Use the method groupby to verify that the bins have about equal size.

In [ ]:
```
# Discretize the variable time_since_last_donation in 10 bins
basetable["bins_recency"] = pd.qcut(basetable["time_since_last_donation"], 10)

# Print the group sizes
print(basetable.groupby("bins_recency").size())
```

```
bins_recency
[32, 319]       10058
(319, 462]       9953
(462, 574]       9999
(574, 657]      10070
(657, 738]      10002
(738, 833]       9994
(833, 933]       9962
(933, 1050]     10009
(1050, 1209]    10004
(1209, 2518]     9949
dtype: int64
```

The variable is binned in 10 bins that have each about 10 000 observations in it!

Discretizing all variables

Instead of discretizing the continuous variables one by one, it is easier to discretize them automatically. To get a list of all the columns in Python, you can use

variables = basetable.columns Only variables that are continuous should be discretized. You can verify whether variables should be discretized by checking whether they have more than a predefined number of different values.

Instructions 100 XP Make a list variables containing all the column names of the basetable. Create a loop that checks all the variables in the list variables. Complete the ifstatement such that only variables with more than 5 different values are discretized. Group the continuous variables in 10 bins using the qcut method.

In [ ]:
```
# Print the columns in the original basetable
print(basetable.columns)

# Get all the variable names except "target"
variables = list(basetable.columns)
variables.remove("target")

# Loop through all the variables and discretize in 10 bins if there are more than 5 different values
for variable in variables:
    if len(basetable.groupby(variable))>5:
        new_variable = "disc_" + variable
        basetable[new_variable] = pd.qcut(basetable[variable], 10)

# Print the columns in the new basetable
print(basetable.columns)
```

```
Index(['target', 'gender_F', 'gender_M', 'income_average', 'income_low',
       'income_high', 'country_USA', 'country_India', 'country_UK', 'age',
       'time_since_last_gift', 'time_since_first_gift', 'max_gift', 'min_gift',
       'mean_gift', 'median_gift'],
      dtype='object')
Index(['target', 'gender_F', 'gender_M', 'income_average', 'income_low',
       'income_high', 'country_USA', 'country_India', 'country_UK', 'age',
       'time_since_last_gift', 'time_since_first_gift', 'max_gift', 'min_gift',
       'mean_gift', 'median_gift', 'disc_age', 'disc_time_since_last_gift',
       'disc_time_since_first_gift', 'disc_max_gift', 'disc_min_gift',
       'disc_mean_gift', 'disc_median_gift'],
      dtype='object')
```

Making clean cuts

The qcut method divides the variable in n_bins equal bins. In some cases, however, it is nice to choose your own bins. The method cut in python allows you to choose your own bins.

Instructions 100 XP Discretize the variable number_gift in three bins with borders 0 and 5, 5 and 10, 10 and 20 and assign this variable to a new column called disc_number_gift. Count the number of observations in each group.

In [ ]:
```
# Discretize the variable and assign it to basetable["disc_number_gift"]
basetable["disc_number_gift"] = pd.cut(basetable["number_gift"], [0,5,10,20])

# Count the number of observations per group
print(basetable.groupby("disc_number_gift").size())
```

```
disc_number_gift
(0, 5]      55063
(5, 10]     41120
(10, 20]     3817
dtype: int64
```

Calculating average incidences

The most important column in the predictor insight graph table is the target incidence column. This column shows the average target value for each group.

Instructions 100 XP Create a dataframe basetable_income that only contains the variables target and income. Group this dataframe by income. Calculate the average target incidence for each group in income.

In [ ]:
```
# Select the income and target columns
basetable_income = basetable[["target","income"]]

# Group basetable_income by income
groups = basetable_income.groupby("income")

# Calculate the target incidence and print the result
incidence = groups["target"].agg({"Incidence" : np.mean}).reset_index()
print(incidence)
```

```
      income  Incidence
0  average   0.049166
1     high   0.061543
2      low   0.043118
```

You can observe that the higher a donor's income, the more likely he is to donate for the campaign.

Constructing the predictor insight graph table

In the previous exercise you learned how to calculate the incidence column of the predictor insight graph table. In this exercise, you will also add the size of the groups, and wrap everything in a function that returns the predictor insight graph table for a given variable.

Instructions 100 XP Group the basetable by variable. Calculate the predictor insight graph table by calculating the target incidence and group sizes. Use the function create_pig_table to calculate the predictor insight graph table for the variable "gender".

In [ ]:
```
# Function that creates predictor insight graph table
def create_pig_table(basetable, target, variable):

    # Create groups for each variable
    groups = basetable[[target,variable]].groupby(variable)

    # Calculate size and target incidence for each group
    pig_table = groups[target].agg({'Incidence' : np.mean, 'Size' : np.size}).reset_index()

    # Return the predictor insight graph table
    return pig_table

# Calculate the predictor insight graph table for the variable gender
pig_table_gender = create_pig_table(basetable, "target","gender")

# Print the result
print(pig_table_gender)
```

```
  gender  Incidence   Size
0      F   0.053844  50033
1      M   0.045970  49967
```

Grouping all predictor insight graph tables

In the previous exercise, you constructed a function that calculates the predictor insight graph table for a given variable as follows:

pig_table = create_pig_table(basetable, "target","variable") If you want to calculate the predictor insight graph table for many variables at once, it is a good idea to store them in a dictionary. You can create a new dictionary using dictionary = {}, add elements with a key using dictionary["key"] = value and retrieve elements using the key print(dictionary["key"]).

Instructions 100 XP Create an empty dictionary pig_tables. For each variable, create a predictor insight graph table. For each variable, add this predictor insight graph table to the dictionary, with as key the name of the variable. Print the predictor insight graph table of disc_time_since_last_gift.

In [ ]:
```
# Variables you want to make predictor insight graph tables for
variables = ["income","gender","disc_mean_gift","disc_time_since_last_gift"]

# Create an empty dictionary
pig_tables = {}

# Loop through the variables
for variable in variables:

    # Create a predictor insight graph table
    pig_table = create_pig_table(basetable, "target",variable)

    # Add the table to the dictionary
    pig_tables[variable] = pig_table

# Print the predictor insight graph table of the variable "disc_time_since_last_gift"
print(pig_tables["disc_time_since_last_gift"])
```

```
  disc_time_since_last_gift  Incidence   Size
0              (1050, 2518]   0.023255  19953
1                (462, 657]   0.061986  20069
2                (657, 833]   0.050810  19996
3               (833, 1050]   0.033799  19971
4                 [32, 462]   0.079556  20011
```

The predictor insight graph table shows that people who donated recently are more likely to donate again.

Plotting the incidences

The most important element of the predictor insight graph are the incidence values. For each group in the population with respect to a given variable, the incidence values reflect the percentage of targets in that group. In this exercise, you will write a python function that plots the incidence values of a variable, given the predictor insight graph table.

Instructions 1/2 50 XP 1 2 Select the right column in the predictor insight graph table and plot this column.

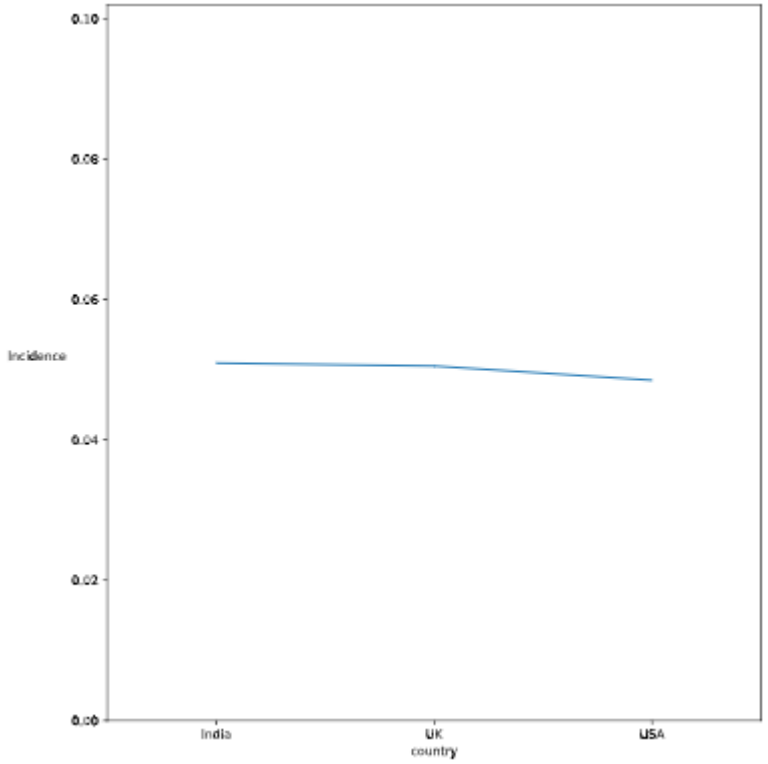
In [ ]:
```python
# The function to plot a predictor insight graph.
def plot_incidence(pig_table, variable):

    # Plot the incidence line
    pig_table["Incidence"].plot()

    # Add Labels to the horizontal axis
    plt.xticks(np.arange(len(pig_table)), pig_table[variable])
    plt.xlim([-0.5, len(pig_table) - 0.5])
    plt.ylim([0, max(pig_table["Incidence"] * 2)])
    plt.ylabel("Incidence", rotation=0, rotation_mode="anchor", ha="right")
    plt.xlabel(variable)

    # Show the graph
    plt.show()
```

Plotting the incidences

The most important element of the predictor insight graph are the incidence values. For each group in the population with respect to a given variable, the incidence values reflect the percentage of targets in that group. In this exercise, you will write a python function that plots the incidence values of a variable, given the predictor insight graph table.

Instructions 2/2 50 XP 2 The predictor insight graph table is given in pig_table. Use the function plot_incidence to show the predictor insight graph of the variable "country".

In [ ]:
```python
# The function to plot a predictor insight graph.
def plot_incidence(pig_table, variable):

    # Plot the incidence line
    pig_table["Incidence"].plot()

    # Formatting the predictor insight graph
    plt.xticks(np.arange(len(pig_table)), pig_table[variable])
    plt.xlim([-0.5, len(pig_table) - 0.5])
    plt.ylim([0, max(pig_table["Incidence"]*2)])
    plt.ylabel("Incidence", rotation=0, rotation_mode="anchor", ha="right")
    plt.xlabel(variable)

    # Show the graph
    plt.show()

# Apply the function for the variable "country".
plot_incidence(pig_table, "country")
```



Plotting the group sizes

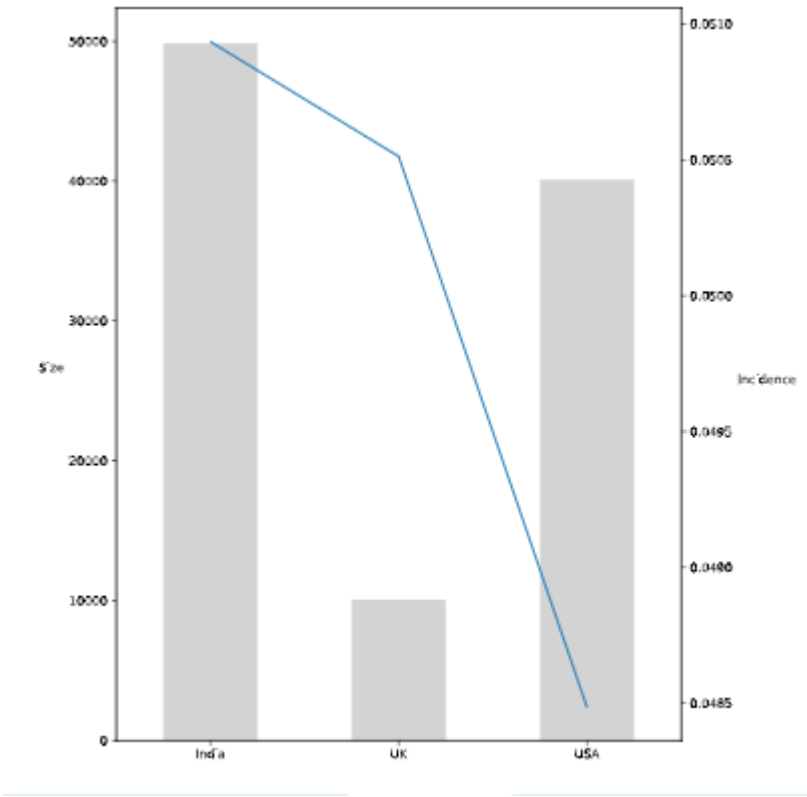
The predictor insight graph gives information about predictive variables. Each variable divides the population in several groups. The predictor insight graph has a line that shows the average target incidence for each group, and a bar that shows the group sizes. In this exercise, you will learn how to write and apply a function that plots a predictor insight graph, given a predictor insight graph table.

Instructions 1/2 50 XP 1 2 Plot the bars that show the Size of each group. Plot the incidence line that shows the average target incidence of each group.

In [ ]:
```
# The function to plot a predictor insight graph
def plot_pig(pig_table, variable):

    # Plot formatting
    plt.ylabel("Size", rotation=0, rotation_mode="anchor", ha="right")

    # Plot the bars with sizes
    pig_table["Size"].plot(kind="bar", width=0.5, color="lightgray", edgecolor="none")

    # Plot the incidence line on secondary axis
    pig_table["Incidence"].plot(secondary_y=True)

    # Plot formatting
    plt.xticks(np.arange(len(pig_table)), pig_table[variable])
    plt.xlim([-0.5, len(pig_table) - 0.5])
    plt.ylabel("Incidence", rotation=0, rotation_mode="anchor", ha="left")

    # Show the graph
    plt.show()
```

In [ ]:

```
# The function to plot a predictor insight graph
def plot_pig(pig_table, variable):

    # Plot formatting
    plt.ylabel("Size", rotation=0, rotation_mode="anchor", ha="right")

    # Plot the bars with sizes
    pig_table["Size"].plot(kind="bar", width=0.5, color="lightgray", edgecolor="none")

    # Plot the incidence line on secondary axis
    pig_table["Incidence"].plot(secondary_y=True)

    # Plot formatting
    plt.xticks(np.arange(len(pig_table)), pig_table[variable])
    plt.xlim([-0.5, len(pig_table) - 0.5])
    plt.ylabel("Incidence", rotation=0, rotation_mode="anchor", ha="left")

    # Show the graph
    plt.show()

# Apply the function for the variable "country"
plot_pig(pig_table, "country")
```



Putting it all together

In the previous exercises, you defined a function create_pig_table that, given a basetable and a predictor, creates a predictor insight graph table for this predictor:

pig_table = create_pig_table(basetable,target,variable) Next, you wrote a function plot_pig that plots the predictor insight graph based on this predictor insight graph table

plot_pig(pig_table, variable) Often, you want to make many predictor insight graphs at once. In this exercise, you will learn how to automatically do this using a for loop.

Instructions 100 XP For each variable in variables, create a predictor insight graph table. The basetable is loaded in basetable. For each variable in variables, plot the predictor insight graph.

In [ ]:
```
# Variables you want to make predictor insight graph tables for.
variables = ["income","gender","disc_mean_gift","disc_time_since_last_gift"]

# Loop through the variables
for variable in variables:

    # Create the predictor insight graph table
    pig_table = create_pig_table(basetable, "target",variable)

    # Plot the predictor insight graph
    plot_pig(pig_table,variable)
```

In [ ]:
```
The above will give several graphs
```