Predict Greenweez Churners

In this challenge were are going to look at the Greenweez client database!

We want to find out who amongst our existing database of clients will reconvert 🔄 (ie. make a second purchase) within 3 months.

Our data

➡️ We have access to the sales from 2019 to 2021.

➡️ Let's take a look

Data exploration

a) Execute the cell below to load our client data into a dataframe variable called `df`.

```
from google.colab import auth
import pandas as pd

# Will collect your credentials
auth.authenticate_user()

# Query Bigquery
query = "SELECT * FROM `data-analytics-bootcamp-363212.course33.gwz_churn`"
project = "data-analytics-bootcamp-363212"

df = pd.read_gbq(query=query, project_id=project)
```

b) Let's take a look at our data.

1. As usual, it's useful to first look at the first few rows.
2. What's the shape of our data?
3. Are there any null values?
4. Given that we are trying to predict 'reconversions', what is our target?

```
df.head()
```

	date_date	orders_id	customers_id	nb_past_orders	avg_basket	total_purchase_cost	avg_quantity	total_quantity	nb_da
0	2021-03-08	797405	207754	3	65.456667	196.37	29.333333	88	
1	2021-06-23	914331	229390	2	84.650000	169.30	40.000000	80	
2	2021-04-27	857750	4921	3	48.343333	145.03	20.000000	60	
3	2021-02-28	786589	10797	8	74.970000	599.76	26.500000	212	
4	2021-06-08	901782	116681	3	62.113333	186.34	16.666667	50	

```
df.shape[1]
```

```
12
```

```
df.isnull().sum()
```

```
date_date              0
orders_id              0
customers_id           0
nb_past_orders         0
avg_basket             0
total_purchase_cost    0
avg_quantity           0
total_quantity         0
nb_days_since_last_order    0
```

```
avg_nb_unique_products      0
total_nb_codes              0
re_purchase                 0
dtype: int64
```

▶ *Answer:*

c) What do you think of column `orders_id` for our problem? Is it useful for our analysis?

▶ *Answer:*

d) Now, delete `orders_id` and `date_date` columns.

```
df = df.drop(['orders_id','date_date'],axis=1)
```

e) Have a look at whether our columns values are on different scales. To do this, use the DataFrame `.describe()` method to compare them.

What kind of preprocessing we will have to use ?

```
df.describe()
```

	customers_id	nb_past_orders	avg_basket	total_purchase_cost	avg_quantity	total_quantity	nb_days_since_last_or
count	381398.0	381398.0	381398.000000	381398.000000	381398.000000	381398.0	3813
mean	161066.560242	2.058692	51.570302	124.525402	13.558555	33.825301	
std	95853.282456	2.030991	41.144718	291.427518	13.202761	71.181359	
min	2.0	1.0	0.000000	0.000000	1.000000	1.0	
25%	69762.0	1.0	26.290000	30.310000	5.500000	7.0	
50%	174880.0	1.0	43.760000	65.550000	10.333333	16.0	
75%	244394.0	2.0	66.840000	150.350000	18.000000	38.0	
max	314334.0	61.0	4726.440000	22738.110000	1480.000000	3557.0	

▶ *Answer:*

f) Set column `customers_id` as index to keep customer_id information.

```
df.set_index('customers_id')
```

	nb_past_orders	avg_basket	total_purchase_cost	avg_quantity	total_quantity	nb_days_since_last_order	avg
customers_id							
207754	3	65.456667	196.37	29.333333	88	0	
229390	2	84.650000	169.30	40.000000	80	0	
4921	3	48.343333	145.03	20.000000	60	0	
10797	8	74.970000	599.76	26.500000	212	0	
116681	3	62.113333	186.34	16.666667	50	0	
...	...	...	...	...	...	...	
114963	4	121.277500	485.11	11.250000	45	0	
28623	2	83.235000	166.47	22.500000	45	0	
185134	2	33.470000	66.94	22.500000	45	0	
230254	2	28.180000	56.36	22.500000	45	0	
206446	4	118.770000	475.08	11.250000	45	0	

381398 rows × 9 columns

Note that for the sake of the exercise, we've **already preprocessed some of the data for you** 🔧.

This means you'll be working on a (relatively) clean database, with your targets and features already formed. In a real-world situation, it's likely that you'll be spending a lot of time forming your target and features from simpler, less-specific data, either using python or SQL to manipulate the database.

Modeling

Now that we've seen what our data looks like, we need to define our target and features.

a) Split dataset into a train and a test set (this should give you an `X_train`, `X_test`, `y_train` and `y_test`).

We will keep a test_size of 20%.

```
from sklearn.model_selection import train_test_split

# every columns in X variable except re_purchase which is our target
X = df[['nb_past_orders','avg_basket','total_purchase_cost','avg_quantity','total_quantity','nb_days_since_last_order','avg_
y = df['re_purchase']

# split data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# store customers_ids for after
test_customers_ids = X_test.index
```

b) Execute the cell below to apply normalization on our data. We are going to use a StandardScaler for this transformation.

Make sure you understand what this code does.

Why do we use `.fit_transform()` on the train set and `.transform()` on the test set?

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

# apply same transformation on X_test
X_test = scaler.transform(X_test)
```

c) What are the types of X_train and X_test?

```
# print(type(X_train))
print(type(X_test))
```

```
<class 'numpy.ndarray'>
```

d) Before building our first model we need a baseline to compare our futur models!

For this example, calculate the accuracy score for a stupid model returning always 1.

```
import numpy as np
from sklearn.metrics import accuracy_score

baseline_y_pred = pd.Series(np.ones([76280]))

baseline_accuracy = accuracy_score(y_test, baseline_y_pred)

print(f"Baseline accuracy is {round(baseline_accuracy,2)}")
```

```
Baseline accuracy is 0.48
```

Now that we have a baseline, even if it's poor, we will try to surpass it!

e) Let's build our first model!

We will use a simple logistic regression model. Execute cell below to train your model on the train data and store the test data predictions in a variable `y_pred`.

Make sure you understand what this code does.

```
from sklearn.linear_model import LogisticRegression

# train model
clf = LogisticRegression()
clf.fit(X_train, y_train)

# store predictions
y_pred = clf.predict(X_test)
```

f) Calculate the accuracy you get on test data.

How do you interpret this value?

```
clf.score(X_test, y_test)
```

```
0.7318956476140535
```

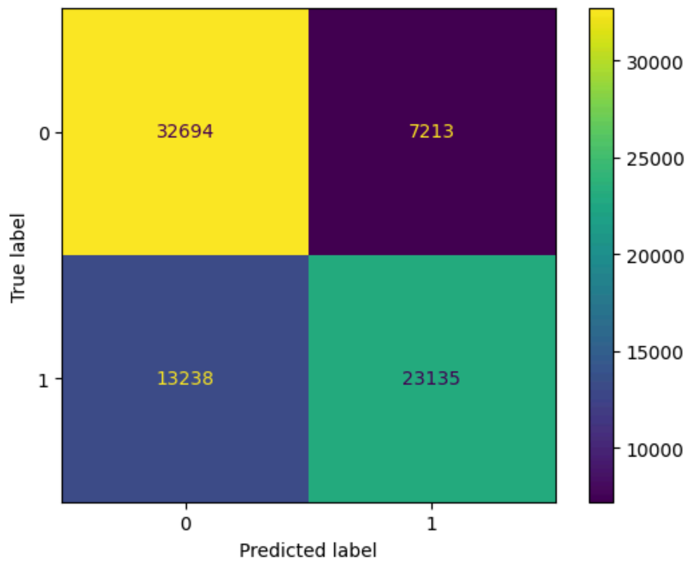
▶ *Answer:*

g) Accuracy is one way to judge model performance, but plotting a confusion matrix on the test data can be more informative. This is because you can calculate additional metrics from this matrix!

Execute cell below to plot confusion matrix.

```
from sklearn.metrics import ConfusionMatrixDisplay

confusion_matrix = ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
```



> precision = tp / tp + fp SO 32694 / 32694 + 13238

> recall = tp / tp + fn SO 32694 / 32694 + 7213

> accuracy = tp + tn / total_count

h) From your confusion matrix and the below picture, calculate:

- precision
- recall
- accuracy

Remember to make sure you are using the values associated with the correct labels when doing so!

confusionmetrucs

▶ *Answer:*

i) What is the percent of churners your model correctly detected? Is it good?

churners = TP / TP + FN = 82%

▶ *Answer:*

j) What does the code below do? Why would this be useful from a business perspective?

> very useful to see which customers to focus on, pushing both in the not churn direction

```
proba = pd.DataFrame(clf.predict_proba(X_test), columns=["Churner", "Not churner"], index=test_customers_ids)
proba
```

	Churner	Not churner
76891	0.466961	0.533039
119294	0.782320	0.217680
374380	0.201453	0.798547
300187	0.233103	0.766897
221619	0.752548	0.247452
...	...	...
335642	0.794768	0.205232
83809	0.416544	0.583456
90416	0.794345	0.205655
48369	0.159639	0.840361
239986	0.595399	0.404601

76280 rows × 2 columns

▶ *Answer:*

k) Filter this dataframe on customers who have between 20% and 50% probability to re purchase.

Customers with a probability of less than 20% to repurchase are considered lost.

```
proba_no_churn = proba[(proba['Not churner'] <= 0.5) & (proba['Not churner'] > 0.2)]
proba_no_churn
```

	Churner	Not churner
119294	0.782320	0.217680
221619	0.752548	0.247452
199063	0.774402	0.225598
175808	0.761864	0.238136
228614	0.750030	0.249970
...	...	...
274354	0.708104	0.291896
348413	0.786766	0.213234
335642	0.794768	0.205232
90416	0.794345	0.205655
239986	0.595399	0.404601

44844 rows × 2 columns

l) Well done! You now have a model that predicts churners.

Using this model, suggest a process that can be implemented at GreenWeez to help the company reduce the churn rate.

▼ *Answer:*

On a regular basis, predict possible churners and send this info back to the CRM via ELT.

The CRM team will then target those users who are predicted as likely to churn by sending them coupon codes, discounts, ... things that will hopefully retain them!