Customer churn is the percentage of customers that stopped using the company’s product or service during a given time period, and it's one of the most important metrics for businesses to evaluate. Customer churn is a critical metric because it is much less expensive to retain existing customers than it's to acquire new ones.

A model that can get a good prediction whether a customer is going to leave or not, could be a valuable asset to companies. The companies could improve their customer retention by knowing on whitch customers they need to put their afffort to preserve. In this notebook I built this kind of model using the "Telcom Customer Churn" dataset. Telcom companies experiencing a lot of fluctuation in their customers and this model can be very usfull to this type of companies.

In addition, I will demonstrate the importance of balancing the train data (in case that is imbalanced). When dealing with churn data, useally the churn precantage will be very low (if it's not, it's a much bigger problem that even this model won't help..)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
import xgboost as xgb
```

Importing the data:

```
data = pd.read_csv('telecom_churn.csv')
```

```
data.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	Nc
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	Nc

We can see that for each customer we have some demographic info, the services he uses and information about his contract.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64 
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64 
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

We can see that we don't have any nulls, great! But the type of feature "TotalCharges" suppose to be 'float64', not 'object', because it's numerical. I will check why:

```
data.TotalCharges = pd.to_numeric(data.TotalCharges)
```

The feature "TotalCharges" is a product of the features "tenure" * "MonthlyCharges". In the sampels where "tenure" = 0, there was suppose to be "TotalCharges" = 0, but instead there is just a space " ". I will fix it.

```
data = data.replace(' ', value=0)
data.TotalCharges = pd.to_numeric(data.TotalCharges)
data.TotalCharges.dtype
```

Now I will check if the data is indeed imbalanced:

```
g = sns.countplot(x="Churn",data=data, palette="muted")
g.set_ylabel("Customers", fontsize=14)
```

```
g.set_xlabel("Churn", fontsize=14)

data.Churn.value_counts(normalize=True)
```

The churn customers are only a quarter from the data. Later I will balanced it to get a better model.

Preprocessing:

First of all, I will arange the data so the algorithm could work with it:

The first column, "customerID", is unnecessary so I will drop it.

```
data = data.drop('customerID', axis = 1)
```

Let's check what kind of catagorical data do we have:

```
catagorical = [i for i in data.columns if data[i].dtypes == 'object']

for i in catagorical:
    print(i, ':', data[i].unique())
```

We have a lot of "yes\no" features (also in the "Churn" column), I will change them and the "gender" feature (male\female) to 0 and 1. Before that I will change the 'No phone/internet service' to 'No', because it's not adding any new informaion (we have these information in the "PhoneService" and "InternetService" features).

```
data = data.replace(regex=r'No\s[a-z]+\sservice', value='No')

for i in catagorical:
    if len(data[i].unique()) == 2:
        data[i] = data[i].map({'Male': 0, 'Female': 1, 'No': 0, 'Yes': 1})
```

The catagorical features that remained after the changes:

```
catagorical = [i for i in data.columns if data[i].dtypes == 'object']

for i in catagorical:
    print(i, ':', data[i].unique())
```

I will check if some of them can be considerd as ordinal:

```
data = data.replace(regex=r'\s\(automatic\)', value='')

fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
plt.subplots_adjust(wspace=0.05, hspace=0.05)

g1 = sns.barplot(ax=axes[0], x="InternetService",y="Churn",data=data, palette="mu
g1.set_xlabel("InternetService", fontsize=14)
g1.set_ylabel("Churn probability", fontsize=14)

g2 = sns.barplot(ax=axes[1], x="Contract",y="Churn",data=data, palette="muted")
g2.set_xlabel("Contract", fontsize=14)
g2.set_ylabel('')

g3 = sns.barplot(ax=axes[2], x="PaymentMethod",y="Churn",data=data, palette="mute
g3.set_xlabel("PaymentMethod", fontsize=14)
g3.set_ylabel('')
g3.set_xticklabels(labels=data.PaymentMethod.unique(), fontsize=8)
```

We can clearly see that the features "InternetService" and "Contract" can be considerd ordinals and they have good correlation with the churn labels. So I will change them accordingly.

```
internetservice_keys = data.InternetService.unique()
internetservice_mapping = dict(zip(internetservice_keys, [1, 2, 0]))
data.InternetService = data.InternetService.map(internetservice_mapping)

contract_keys = data.Contract.unique()
contract_mapping = dict(zip(contract_keys, [2, 1, 0]))
data.Contract = data.Contract.map(contract_mapping)
```

```
data.head()
```

Heatmap:

```
corrmat = data.corr()
f, ax = plt.subplots(figsize = (12, 9))
sns.heatmap(corrmat, vmax = 0.8, square = True, cmap = "coolwarm")
```

We can see, like we expected, there is a strong correlation between "tenure" (or "MonthlyCharges") and "TotalCharges" because "TotalCharges" = "tenure" * "MonthlyCharges". In addition, we can see that there is a strong correlation between "OnlineSecurity" and "MonthlyCharges". This is probably because the "fiber optic" is the most expensive service and this is the reason for high monthly charges. It's also can explain why the churn probability is higher for the customers who own this service - the price is too high for them. Another strong negative correlation is between "tenure" and "Contract".It make sense that the higher the tenure - the contract will be yearly and not monthly. Finaly if we look at the stronget correlation with "Churn", it's "tenure". I will check it out:

```
g = sns.lineplot(x="tenure", y="Churn", data=data, palette="muted")
g.set_xlabel("tenure", fontsize=14)
g.set_ylabel("Churn probability", fontsize=14)
```

We can clearly see the trend that the higher the tenure - the lesser chance the customer will leave, make sense. I will fix the anomaly at the beggining where the tenure is 0 - it's a new customer that just joined the compeny services so there is a 0 chance for him to be a churn customer. I will change the 0 tenure to be the maximum tenure so it will fit the trend.

```
mask = data.index[data.tenure == 0]
data.loc[mask, 'tenure'] = data.tenure.max() + 1

g = sns.lineplot(x="tenure", y="Churn", data=data, palette="muted")
g.set_xlabel("tenure", fontsize=14)
g.set_ylabel("Churn probability", fontsize=14)
```

Nice:), it's look like we can divide the "tenure" feature to 3 part:

```
data['tenureBand'] = pd.cut(data.tenure, 3)

g = sns.barplot(x="tenureBand", y="Churn", data=data, palette="muted")
g.set_xlabel("tenureBand", fontsize=14)
g.set_ylabel("Churn probability", fontsize=14)

data = data.drop('tenure', axis = 1)
```

Now I can also change this feature to be ordinal:

```
tenureBand_keys = data.tenureBand.unique()
tenureBand_mapping = dict(zip(tenureBand_keys, [2, 1, 0]))
data.tenureBand = data.tenureBand.map(tenureBand_mapping)

data.tenureBand = pd.to_numeric(data.tenureBand)
```

```
data = pd.get_dummies(data)
data.head()
```

Fitting the model:

```
y = data.Churn.values

x = data.drop('Churn', axis = 1).values
```

Normalizing the data and splitting it to train and test:

```
x = MinMaxScaler().fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=51)
```

To deal with the imbalanced data, I will check two techniques: **1. Over-sampling - SMOTE:** Synthetic Minority Over-sampling Technique SMOTE first selects a minority class instance a at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b. (Page 47, Imbalanced Learning: Foundations, Algorithms, and Applications, 2013.)

2. Under-sampling - NearMiss3: When instances of two different classes are very close to each other, it removes the instances of the majority class to increase the spaces between the two classes. It works in 2 steps: Firstly, for each minority class instance, their M nearest-neighbors will be stored. Then finally, the majority class instances are selected for which the average distance to the N nearest-neighbors is the largest.

```
x_smote, y_smote = SMOTE().fit_resample(x, y)

x_nearmiss, y_nearmiss = NearMiss(version=3).fit_resample(x, y)
```

I will check the results of this 5 classifiers:

```
classifiers = []
classifiers.append(LogisticRegression(max_iter=500))
classifiers.append(RandomForestClassifier())
classifiers.append(AdaBoostClassifier())
classifiers.append(xgb.XGBClassifier())
classifiers.append(SVC(C = 10))
```

I will evaluate this 5 classifiers by using cross validation. This next function finds the best classifier, the classifier who got the best score in the cross validation. I will run it for each of the balancing techniques and without balancing at all.

```
def cross_validate(classifiers, x_train, y_train, x_test, y_test):
    cv_results = []
    for classifier in classifiers:
        cv_results.append((cross_val_score(classifier, x_train, y_train,\
                                           scoring = "f1_weighted")).mean())
    best_clf = classifiers[np.argmax(cv_results)]
    return best_clf

best_clf_wo_balancing = cross_validate(classifiers, x_train, y_train, x_test, y_t
best_clf_smote = cross_validate(classifiers, x_smote, y_smote, x_test, y_test)
best_clf_nearmiss = cross_validate(classifiers, x_nearmiss, y_nearmiss, x_test, y
```

Testing the models:

This function is for printing the classification report on the test data for each of the best classifiers:

```
def print_best_clf_results(best_clf, x_train, y_train, x_test, y_test):

    print(f'\nThe classifier with the best f1 result is: {best_clf}')

    best_clf.fit(x_train, y_train)
    y_pred = best_clf.predict(x_test)

    print('\nClassification Report:')
    print(classification_report(y_test, y_pred))
    print('\n\n')
    f_i = best_clf.feature_importances_
```

```
        return f_i


    print('-'*70)
    print('Without balncing:')
    print('-'*70)
    f_i = print_best_clf_results(best_clf_wo_balancing, x_train, y_train, x_test, y_t

    print('-'*70)
    print('Over-sampling using SMOTE technique:')
    print('-'*70)
    f_i_smote = print_best_clf_results(best_clf_smote, x_smote, y_smote, x_test, y_te

    print('-'*70)
    print('Under-sampling using NearMiss3 technique:')
    print('-'*70)
    f_i_nearmiss = print_best_clf_results(best_clf_nearmiss, x_nearmiss, y_nearmiss,
```

The Random Forest Classifier got an outstanding result on the over-sampling data using SMOTE technique!

We can see that without balancing the data, we got an ok accuracy with AdaBoost algorithm but the precision and recall of the churn customer is very bad. It's means that the model is not reliable and it's not good in recognaizing the churn customer correctly - our goal. This is why it's always importent to balnce the data. The NearMiss technique got an even worst results, it's probably from losing importent data from the under-sampling.

The Random Forest Classifier in combination with SMOTE technique got 100% correct predictions on the test data :)

Let's check which features were the most important for the predictions:

```
    col_names = data.columns.drop('Churn')

    feature_importances = pd.DataFrame({'Feature': col_names, 'Feature Importances':
    feature_importances = feature_importances.sort_values('Feature Importances', asce

    #aligning text to the left:
```

```
feature_importances = feature_importances.style.set_properties(**{'text-align': '
feature_importances = feature_importances.set_table_styles([dict(selector='th', p
feature_importances
```

Like we saw on the heatmap, **the features about the contract - charge, type and tenure, are the ones that affect the most about whether a customer is going to leave or not.**

0s completed at 11:17 PM