# Presentation 04

Objective: generate and present credit values predictions using RFR

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_
```

In [2]:
```
df = pd.read_csv('/home/gmelao/Desktop/default-of-credit-card-clients.csv')
df.columns = df.iloc[0]
df.drop(0, inplace = True)
df.set_index('ID', inplace = True)
pd.set_option('display.max_columns', 24)
pd.set_option('display.max_rows', 24)
```

In [3]:
```
df = df.apply(lambda df: pd.Series(map(float, df)))
```

## Train Test Split

Split arrays or matrices into random train and test subsets.

In [4]:
```
X = df.drop('LIMIT_BAL', axis=1)
y = df['LIMIT_BAL']
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_st
```

## Random Forest Regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

In [6]:
```
model = RandomForestRegressor()
```

In [7]:
```
model.fit(X_train, y_train)
```

Out[7]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [8]:
```
preds = model.predict(X_test)
preds
```

Out[8]:
```
array([ 71300., 180600., 131500., ..., 235100., 168900., 116800.])
```

In [9]:
```
pd.DataFrame(preds, columns=['Credit Prediction'])
```

Out[9]:

| | Credit Prediction |
|---|---|
| **0** | 71300.000000 |
| **1** | 180600.000000 |
| **2** | 131500.000000 |
| **3** | 154300.000000 |
| **4** | 140100.000000 |
| **...** | ... |
| **9895** | 154716.666667 |
| **9896** | 140900.000000 |
| **9897** | 235100.000000 |
| **9898** | 168900.000000 |
| **9899** | 116800.000000 |

9900 rows × 1 columns

## Metrics

MAE: it is the mean of the absolute error. This gives less weight to outliers, which is not sensitive to outliers.

MSE: MSE is a combination measurement of bias and variance of your prediction

RMSE: Take a root of MSE would bring the unit back to actual unit, easy to interpret your model accuracy.

MAPE: MAPE is the sum of the individual absolute errors divided by the demand (each period separately). It is the average of the percentage errors.

In [11]:
```
mae = mean_absolute_error(y_test, preds)
```

In [12]:
```
mse = mean_squared_error(y_test, preds)
```

In [13]:
```
rmse = np.sqrt(mean_squared_error(y_test, preds))
```

In [14]:
```
mape = mean_absolute_percentage_error(y_test, preds)
```

In [15]:
```
print(f'MAE:  {mae}')
print(f'MSE:  {mse}')
print(f'RMSE: {rmse}')
print(f'MAPE: {mape}')
```

```
MAE:  68484.06770094384
MSE:  9025328793.808342
RMSE: 95001.7304779673
MAPE: 0.7853820680953061
```

In [ ]:

In [17]:
```
y_train.mean()
```

Out[17]:
```
167197.61194029852
```

In [20]:
```
baseline = np.arange(9900)
baseline.fill(y_train.mean())
```

In [21]:
```
mae_baseline = mean_absolute_error(y_test, baseline)
mse_baseline = mean_squared_error(y_test, baseline)
rmse_baseline = np.sqrt(mean_squared_error(y_test, baseline))
mape_baseline = mean_absolute_percentage_error(y_test, baseline)
```

In [22]:
```
print(f'MAE:  {mae_baseline}')
print(f'MSE:  {mse_baseline}')
print(f'RMSE: {rmse_baseline}')
print(f'MAPE: {mape_baseline}')
```

```
MAE:  105694.56606060606
MSE:  17081185206.624243
RMSE: 130695.00834624191
MAPE: 1.669181588976168
```

In [ ]:
```
```

In [25]:
```
print(f'MAE / MAE_BASELINE:   {mae_baseline/mae}')
print(f'MSE / MSE_BASELINE:   {mse_baseline/mse}')
print(f'RMSE / RMSE_BASELINE: {rmse_baseline/rmse}')
print(f'MAPE / MAPE_BASELINE: {mape_baseline/mape}')
```

```
MAE / MAE_BASELINE:   1.5433453299262683
MSE / MSE_BASELINE:   1.8925831509143989
RMSE / RMSE_BASELINE: 1.3757118706016893
MAPE / MAPE_BASELINE: 2.1253115608104425
```

In [ ]:
```
```