ML SPRINT

FRAUDULENT CREDIT CARD TRANSACTION DETECTION

UNSUPERVISED USING KNN

Presented by: TEAM FBV

OUR MEMBERS

NAME	REGISTRATION NUMBER
Shivansh Kumar	20BCE11084
Karan Jadhav	20BCE10499
Aman Patel	20BCE10327
Deepak Kumar	20BAI10372

PROBLEM STATEMENT

8

Classification – Unsupervised – KNN – Fraudulent credit card transactions

The dataset we will use contains transactions made by credit cards in September 2013 by European cardholders. The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

The dataset contains 31 columns, only 3 columns make sense which are Time, Amount and Class (fraud or not fraud). If required use PCA to reduce unnecessary dimensions.

WHAT IS FRAUD ?

Fraud is an act of deception used to illegally deprive another person or entity of money, property or legal rights.

"A typical organization loses 5% of their yearly revenues to frauds"

TYPES OF FRAUDS :

1. **Online Fraud**
2. **Credit card Fraud**
3. **Threat**
4. **Theft of Inventory**

Data Science Approaches for Fraud Detection

- Machine Learning (ML)
  - Supervised Learning
    - Data Labeling
    - Unbalanced Data
  - Unsupervised Learning
    - PCA
    - K-Means Clustering
- Deep Neural Network (DNN)

CHALLENGES OF FRAUD DETECTION MODEL

Unbalanced Data

Less than 0.5% credit card transactions are fraud

Operational Efficiency

Less than 8 sec to flag a transaction

Incorrect Flagging

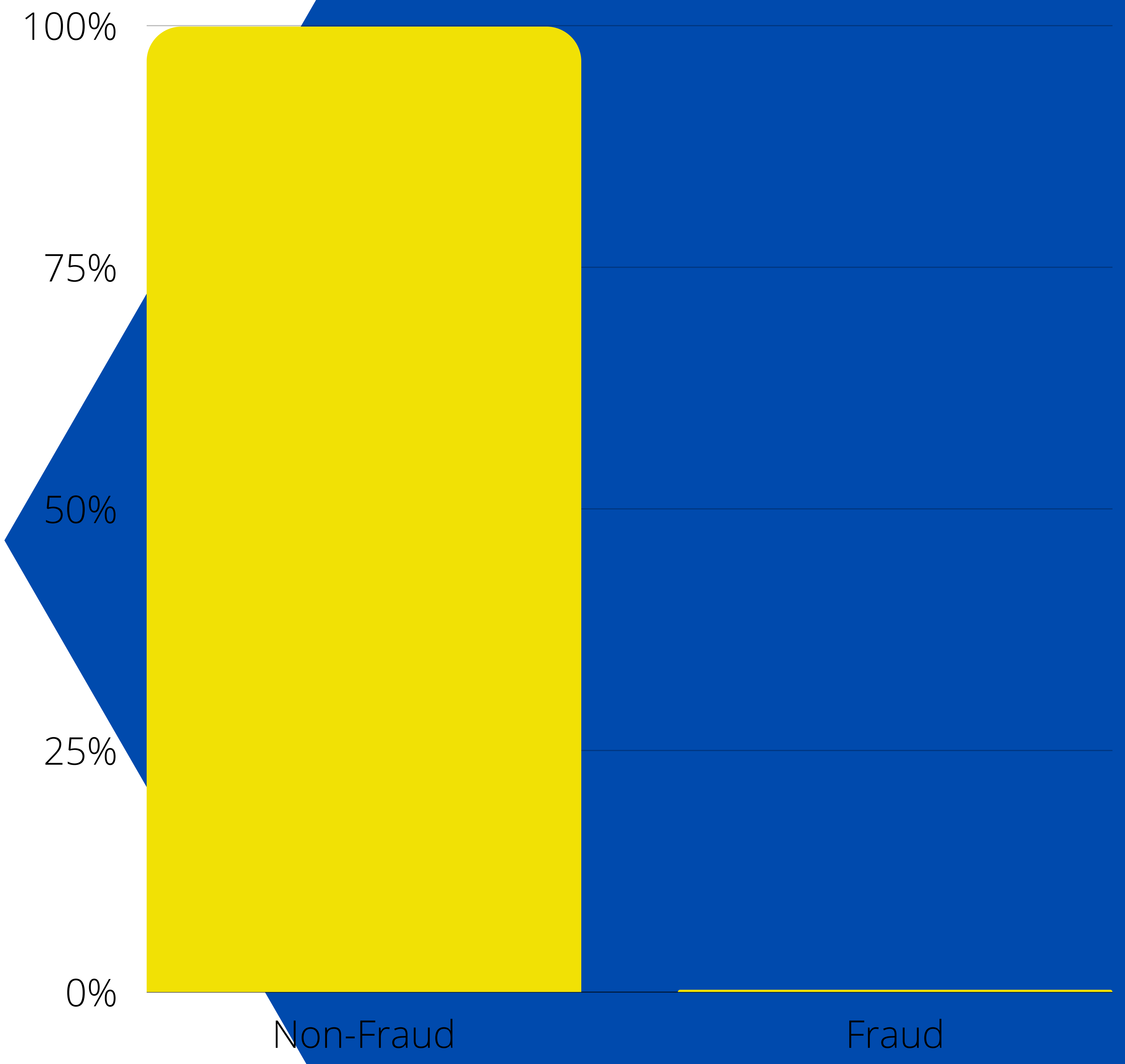
Avoid harassing real customers

Dataset Analysis

Non-Fraud: 284315
Fraud : 492

The ratio of non-fraud to fraud is very high, that is our data is highly imbalanced.

So how to deal with imbalanced data?



DEALING WITH UNBALANCED DATA

01 RANDOM OVER-SAMPLING

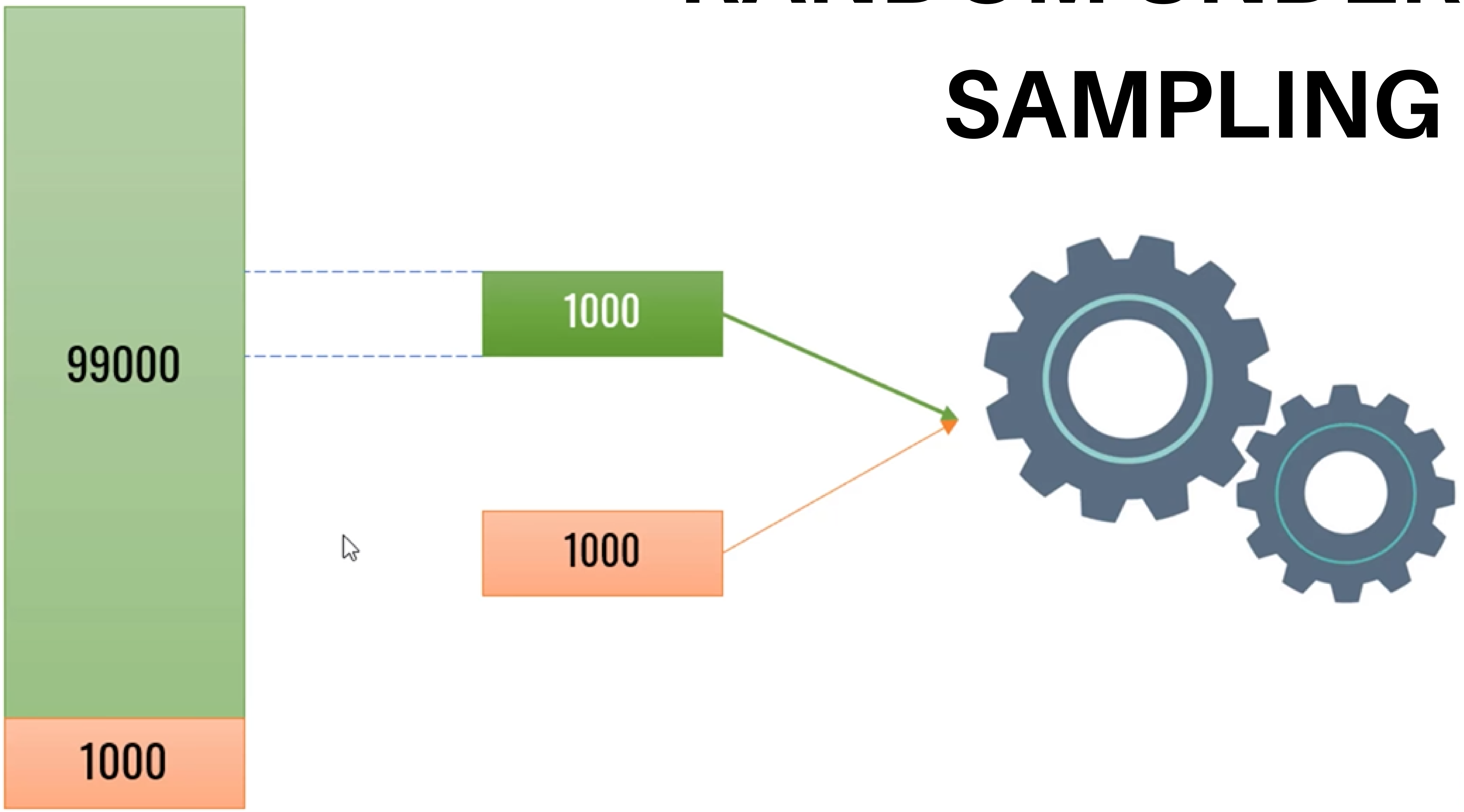
02 RANDOM UNDER-SAMPLING

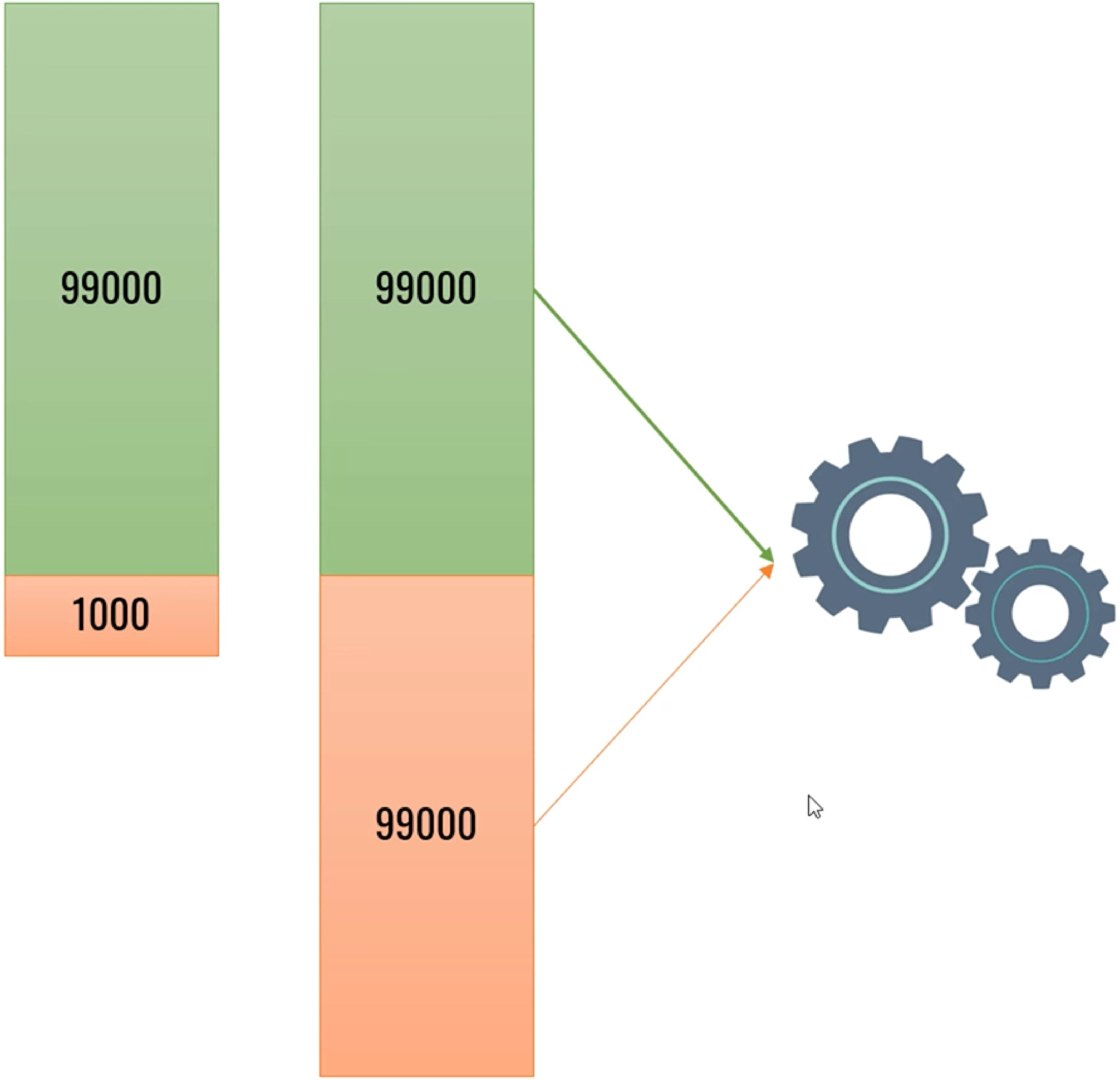
03 OVER SAMPLING MINORITY CLASS USING SMOTE

04 ENSEMBLING

05 FOCAL LOSS

RANDOM UNDER SAMPLING

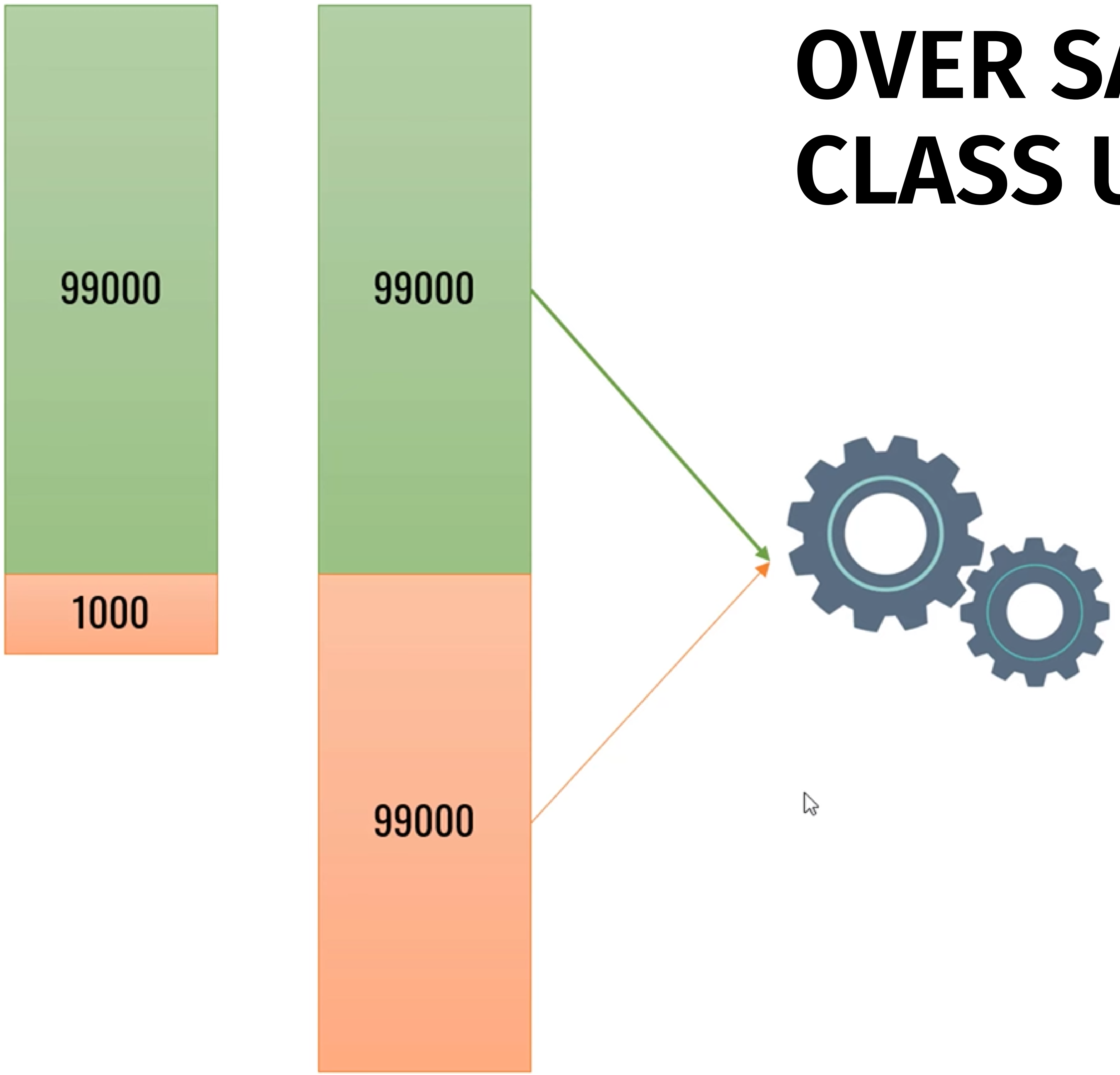




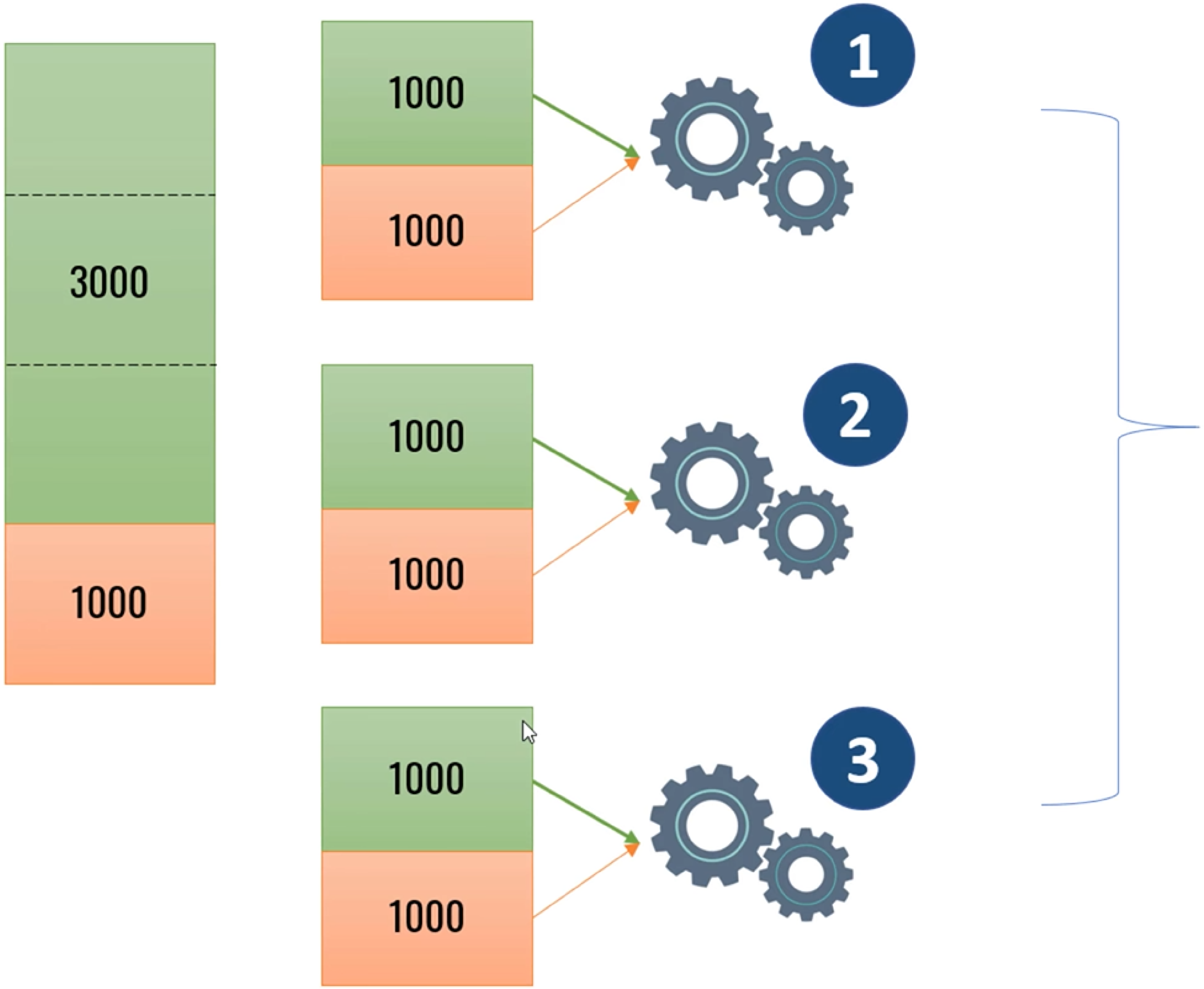
RANDOM OVER SAMPLING

Generate a new sample from the current sample by simply duplicating them

OVER SAMPLING MINORITY CLASS USING SMOTE



1. Generate synthetic examples k nearest neighbors algorithm
2. SMOTE: Synthetic Minority Over-sampling Techinques



Ensemble Method

Majority Vote

OUR APPROCH TO WORK

Understanding Dataset → **Data Wrangling** → **Data Visualization** → **Data Modeling** → **Evaluation**

Understanding Dataset
Imbalanced Data

Data Wrangling
Removed Unnecessary features

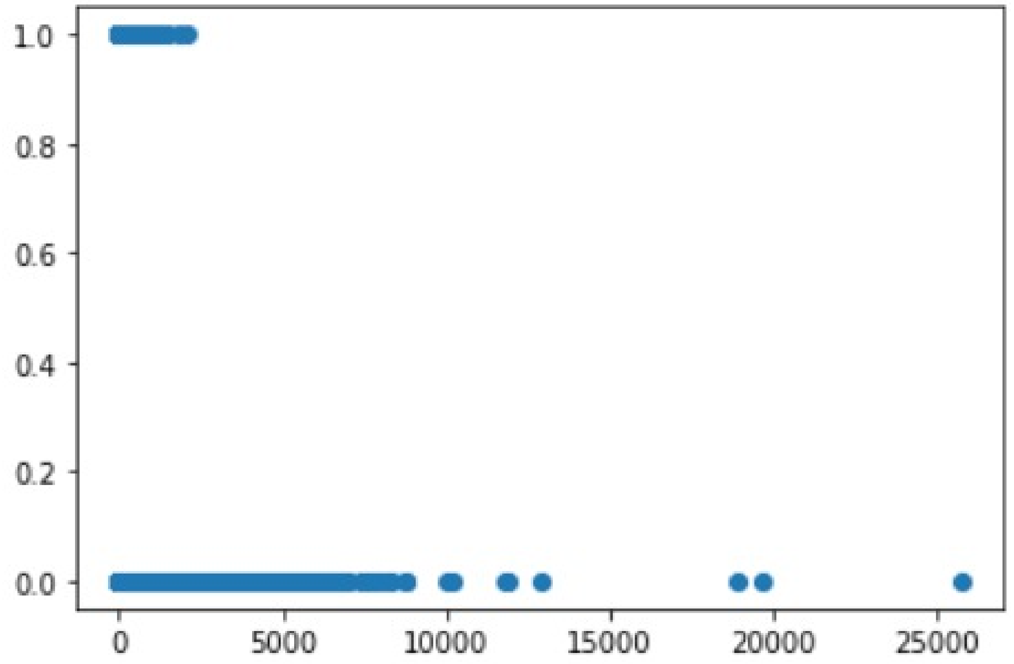
Data Visualization
Understanding data with graphs

Data Modeling
Sampling data and implementation using KNN

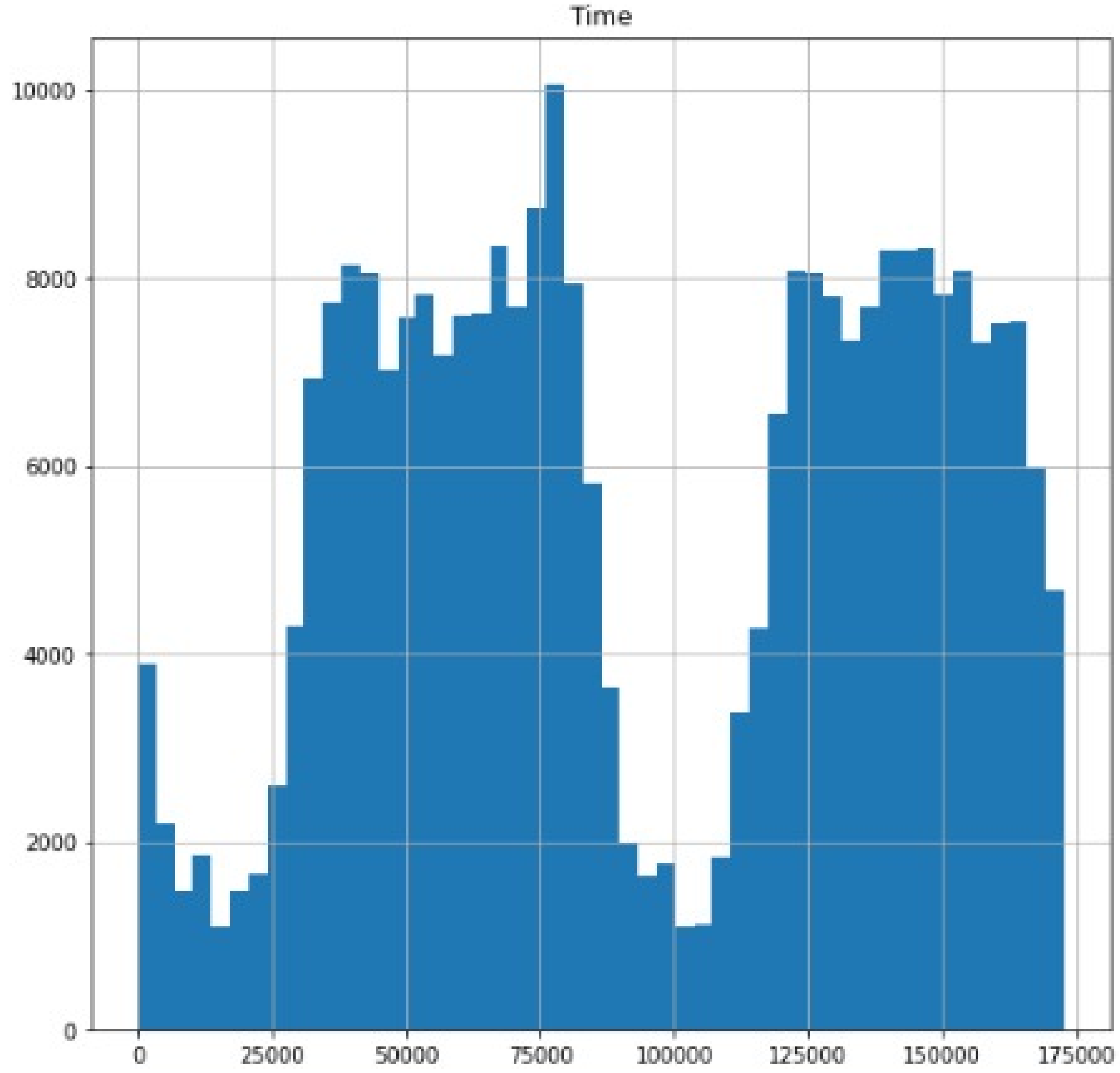
Evaluation
Testing data for best accuracy (recall sccore)

Data Visualization

```
In [56]: plt.scatter(df.Amount, df.Class)
Out[56]: <matplotlib.collections.PathCollection at 0x19dba0e1fa0>
```




```
In [59]: df.Class.value_counts()
Out[59]: 0    284315
         1       492
         Name: Class, dtype: int64
```



Data Models

01 Under Sampling

```
y_undersampling_test.value_counts()
```

```
0    148
1    148
Name: Class, dtype: int64
```

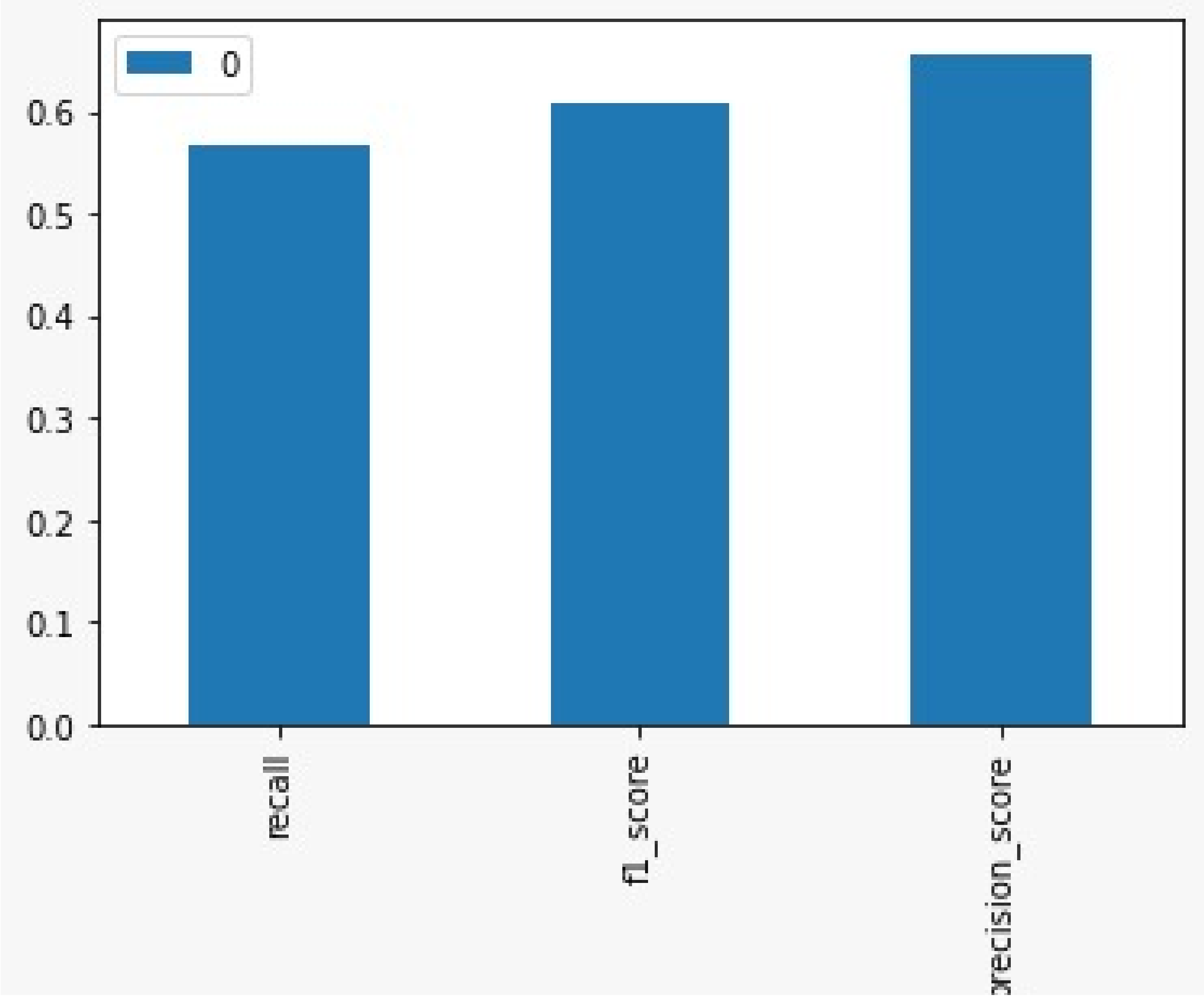
```
knn=KNeighborsClassifier(n_neighbors=3,metric='minkowski',p=2)
knn=knn.fit(x_undersampling_train,np.ravel(y_undersampling_train))
y_pred=knn.predict(x_undersampling_test)
print("kNearest Neighbour")
print("Accuracy")
print(accuracy_score(y_undersampling_test, y_pred))
print(accuracy_score(y_undersampling_test, y_pred,normalize=False))
print("Confusion matrix")
conf_matrix=confusion_matrix(y_undersampling_test,y_pred)
print(conf_matrix)
```

```
kNearest Neighbour
Accuracy
0.6216216216216216
184
Confusion matrix
[[96 52]
 [60 88]]
```

Recall score is important not precision!

```
In [31]: print(classification_report(y_undersampling_test, y_pred))
```

	precision	recall	f1-score	support
0	0.62	0.65	0.63	148
1	0.63	0.59	0.61	148
accuracy			0.62	296
macro avg	0.62	0.62	0.62	296
weighted avg	0.62	0.62	0.62	296



Data Models

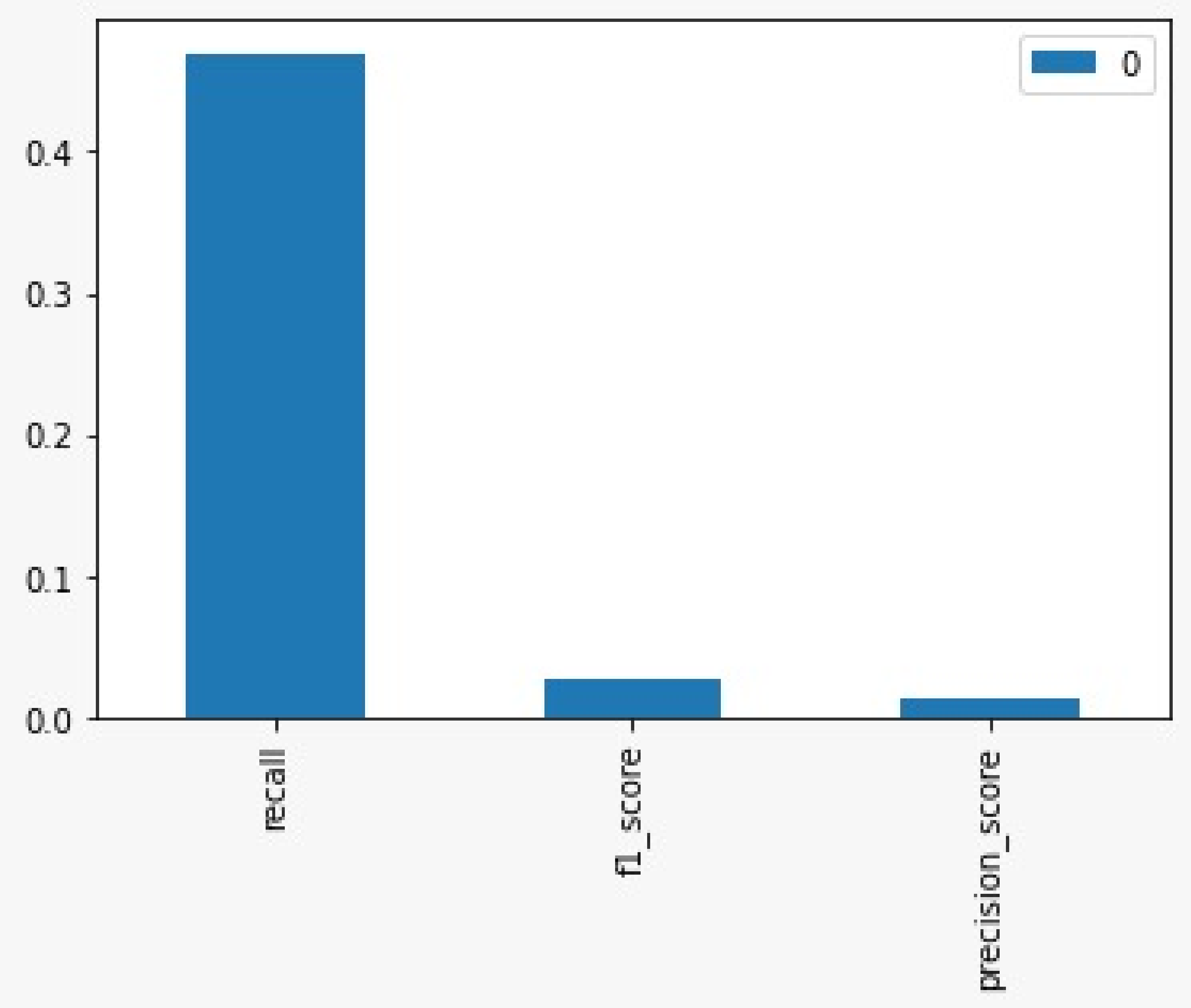
02 Over Sampling - SMOTE

```
In [44]: sm = SMOTE(sampling_strategy=1, random_state=2)
         X_train_large, y_train_large = sm.fit_resample(x_train, y_train)
         X_train_large.size

Out[44]: 279996

In [45]: knn=KNeighborsClassifier(n_neighbors=3,metric='minkowski',p=2)
         knn=knn.fit(X_train_large,y_train_large)
         y_pred=knn.predict(x_test)
         print("kNearest Neighbour")
         print("Accuracy")
         print(accuracy_score(y_test, y_pred))
         print(accuracy_score(y_test, y_pred,normalize=False))
         print("Confusion matrix")
         conf_matrix=confusion_matrix(y_test,y_pred)
         print(conf_matrix)

         kNearest Neighbour
         Accuracy
         0.8308013798593605
         25047
         Confusion matrix
         [[24978  5023]
          [   78    69]]
```

Recall score is important not precision!

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.83	0.91	30001
1	0.01	0.47	0.03	147
accuracy			0.83	30148
macro avg	0.51	0.65	0.47	30148
weighted avg	0.99	0.83	0.90	30148



FBV

Thank you!

ML SPRINT FOR THIS GREAT OPPORTUNITY !