



# Машинное обучение

---

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. Д.Ю. КУПРИЯНОВ

ЛЕКЦИЯ 8 (СЕМЕСТР 2 – 1)

# Библиотеки

---

В данной лекции будут рассмотрены примеры с использованием следующих библиотек:

- NumPy – <https://numpy.org/>
- Pandas – <https://pandas.pydata.org/>
- scikit-learn – <https://scikit-learn.org>
- Matplotlib – <https://matplotlib.org/>

# Часть 1

---

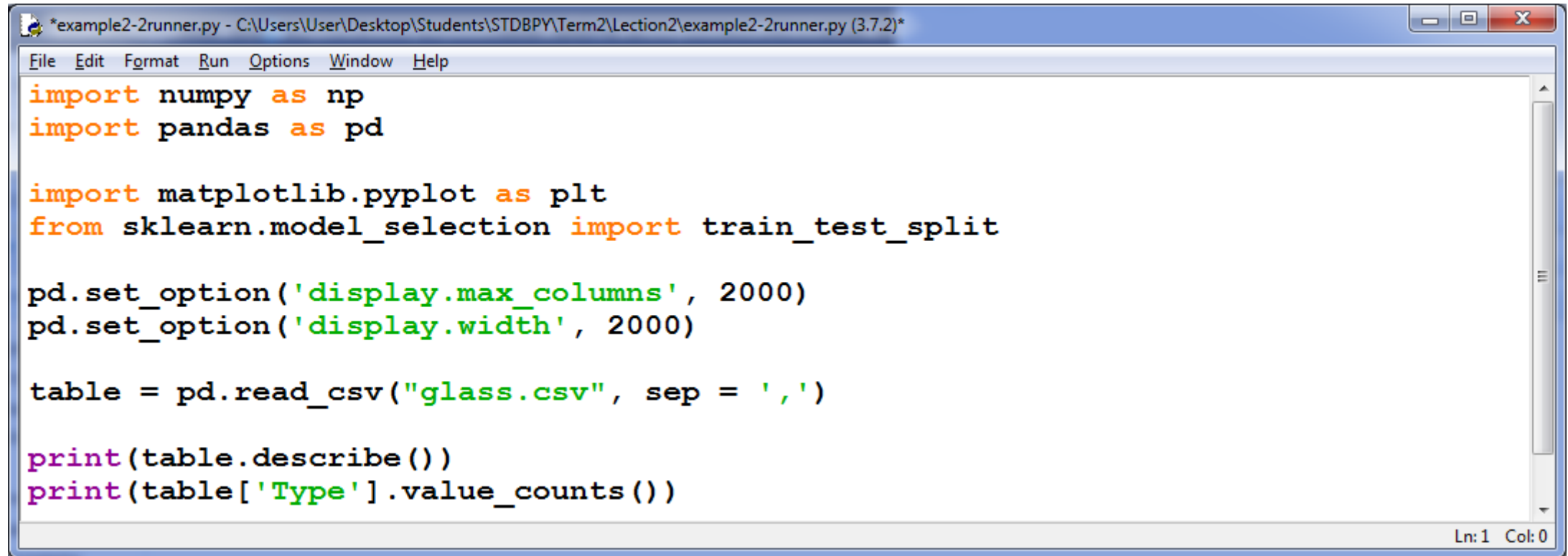
## НЕРАВНОМЕРНАЯ ВЫБОРКА

# Набор данных

---

Загрузим с сайта [kaggle.com](https://www.kaggle.com/uciml/glass) набор данных для классификации стекол:  
<https://www.kaggle.com/uciml/glass>. Изучим его особенности

# Пример 1. Набор данных «Стекло»



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lecion2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)

table = pd.read_csv("glass.csv", sep = ',')

print(table.describe())
print(table['Type'].value_counts())
```

Ln: 1 Col: 0

# Пример 1. Набор данных «Стекло»

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)
table = pd.read_csv("glass.csv", sep = ',')
print(table.describe())
print(table['Type'].value_counts())
```

# Результат

```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\User\Desktop\Students\STDBPY\Term2\Lecture2\example2-2runner.py
count      RI      Na      Mg      Al      Si      K      Ca      Ba      Fe      Type
mean      1.518365  13.407850  2.684533  1.444907  72.650935  0.497056  8.956963  0.175047  0.057009  2.780374
std       0.003037  0.816604  1.442408  0.499270  0.774546  0.652192  1.423153  0.497219  0.097439  2.103739
min       1.511150  10.730000  0.000000  0.290000  69.810000  0.000000  5.430000  0.000000  0.000000  1.000000
25%      1.516523  12.907500  2.115000  1.190000  72.280000  0.122500  8.240000  0.000000  0.000000  1.000000
50%      1.517680  13.300000  3.480000  1.360000  72.790000  0.555000  8.600000  0.000000  0.000000  2.000000
75%      1.519157  13.825000  3.600000  1.630000  73.087500  0.610000  9.172500  0.000000  0.100000  3.000000
max       1.533930  17.380000  4.490000  3.500000  75.410000  6.210000  16.190000  3.150000  0.510000  7.000000
2        76
1        70
7        29
3        17
5        13
6         9
Name: Type, dtype: int64
>>> |
```

Ln: 21 Col: 4

# Недостатки набора данных

---

У набора данных для классификации стекол есть ряд недостатков:

- неравномерное распределение числа наблюдений по классам;
- различие параметров по разбросу значений на порядок и более.

Есть и преимущества:

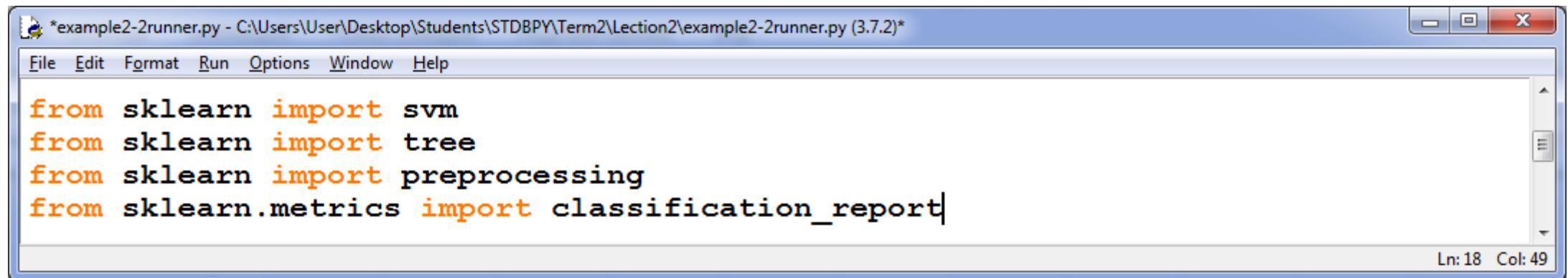
- отсутствие пустых значений;
- отсутствие категориальных признаков.

Второй недостаток мы умеем устранять. Используем для этого стандартизацию (заодно разобьём на тестовую и обучающую выборки)!



# Пример 1. Подключим библиотеки

---



The image shows a screenshot of a Python IDE window. The title bar reads: `*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lecture2\example2-2runner.py (3.7.2)*`. The menu bar includes `File`, `Edit`, `Format`, `Run`, `Options`, `Window`, and `Help`. The main text area contains the following Python code:

```
from sklearn import svm
from sklearn import tree
from sklearn import preprocessing
from sklearn.metrics import classification_report|
```

The status bar at the bottom right indicates `Ln: 18 Col: 49`.

# Пример 1. Стандартизация и разбиение

```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lecture2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(
    table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']])
table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']] = x

train_table, test_table = train_test_split(table, test_size = 0.4,
                                           random_state = 22222)

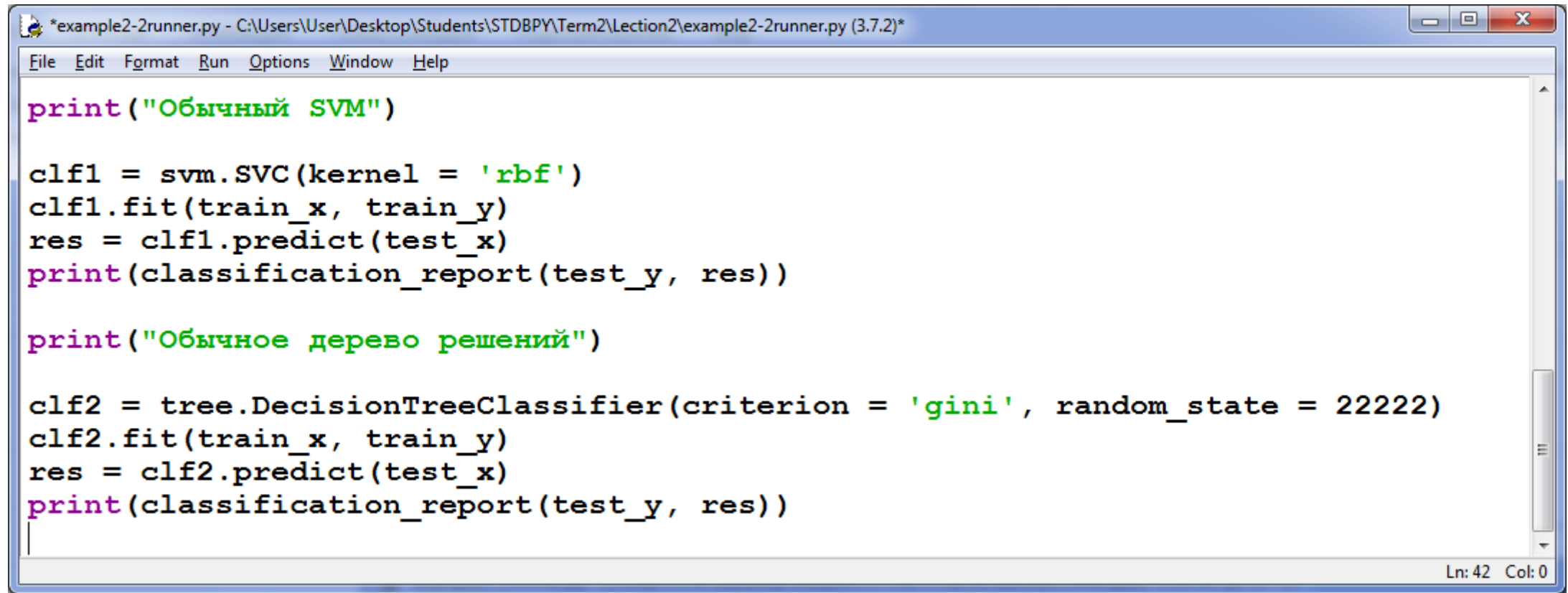
test_table = test_table.reset_index()
train_table = train_table.reset_index()
train_x = train_table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
train_y = train_table['Type']
test_x = test_table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
test_y = test_table['Type']

Ln: 27 Col: 27
```

# Пример 1. Стандартизация и разбиение

```
scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']])
table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']] = x
train_table, test_table = train_test_split(table, test_size = 0.4, random_state = 22222)
test_table = test_table.reset_index()
train_table = train_table.reset_index()
train_x = train_table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']]
train_y = train_table['Type']
test_x = test_table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']]
test_y = test_table['Type']
```

# Пример 1. SVM и DT



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lecture2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

print("Обычный SVM")

clf1 = svm.SVC(kernel = 'rbf')
clf1.fit(train_x, train_y)
res = clf1.predict(test_x)
print(classification_report(test_y, res))

print("Обычное дерево решений")

clf2 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf2.fit(train_x, train_y)
res = clf2.predict(test_x)
print(classification_report(test_y, res))

Ln: 42 Col: 0
```

# Пример 1. SVM и DT

```
print("Обычный SVM")
clf1 = svm.SVC(kernel = 'rbf')
clf1.fit(train_x, train_y)
res = clf1.predict(test_x)
print(classification_report(test_y, res))

print("Обычное дерево решений")
clf2 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf2.fit(train_x, train_y)
res = clf2.predict(test_x)
print(classification_report(test_y, res))
```

# Результат

## Обычный SVM

Warning (from warnings module):

File "C:\Users\User\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\metrics\\_classification.py", line 1268

\_warn\_prf(average, modifier, msg\_start, len(result))

UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

	precision	recall	f1-score	support
1	0.57	0.95	0.71	22
2	0.61	0.63	0.62	35
3	0.00	0.00	0.00	4
5	1.00	0.14	0.25	7
6	0.00	0.00	0.00	3
7	1.00	0.80	0.89	15
accuracy			0.65	86
macro avg	0.53	0.42	0.41	86
weighted avg	0.65	0.65	0.61	86

# Результат

---

Обычное дерево решений					
	precision	recall	f1-score	support	
1	0.65	0.91	0.75	22	
2	0.85	0.66	0.74	35	
3	0.33	0.25	0.29	4	
5	0.56	0.71	0.63	7	
6	0.00	0.00	0.00	3	
7	0.80	0.80	0.80	15	
accuracy			0.71	86	
macro avg	0.53	0.56	0.53	86	
weighted avg	0.71	0.71	0.70	86	

# Был ли данный подход честен?

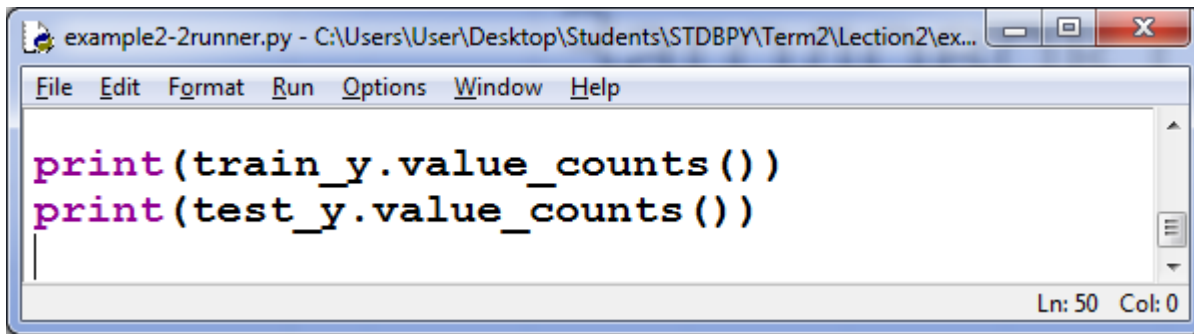
---

Данный результат может показаться хорошим, если не учитывать одной тонкости, при разбиении на тестовую и обучающую выборку мы не учли, что некоторым классам соответствовало мало наблюдений. Если для данных классов в тестовую выборку попадёт мало наблюдений, то мы получим ложный хороший результат.

Посмотрим на тестовую выборку



# Анализ тестовой и обучающей выборок



```
example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lecion2\ex...
File Edit Format Run Options Window Help

print(train_y.value_counts())
print(test_y.value_counts())

Ln: 50 Col: 0
```

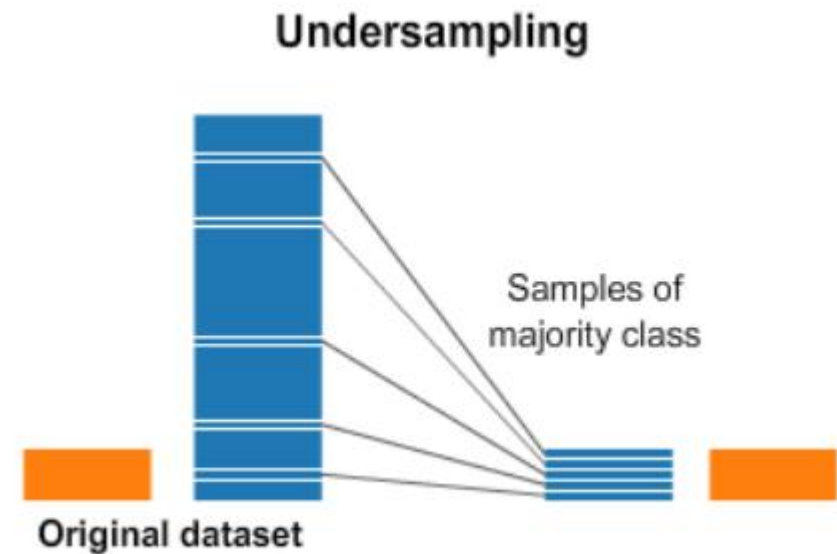
Хотя мы делили выборку в отношении 6 к 4, но для класса 3 реальное деление оказалось 13 к 4, а для класса 5 – 6 к 7. Если в реальном массиве данных, требующем классификации, распределение по классам равновероятно базовой выборке, то качество нашего классификатора будет гораздо хуже, чем мы ожидали. Чтобы избежать данной ситуации при неравномерных классах деление на тестовую и обучающую выборку надо делать на основе стратификации.

```
1      48
2      41
7      14
3      13
6       6
5       6
Name: Type, dtype: int64
2      35
1      22
7      15
5       7
3       4
6       3
Name: Type, dtype: int64
```

# Стратегии балансировки выборки

## Undersampling

**Undersampling** – это такой подход к формированию выборки, когда для каждого класса объектов остается только такое количество строк, которое есть в самом маленьком (по числу строк) классе. Самый простой вариант – это случайный выбор. Но могут быть и более сложные подходы.



# Новая библиотека

---

Для работы с несбалансированными наборами данных может быть очень полезна библиотека Imbalanced Learn. Для установки используйте в консоли команду:

```
путь_к_питону\python -m pip install imbalanced-learn
```



# Undersampling

---

```
example2-2.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture8\example2-2.py (3.11.2)
File Edit Format Run Options Window Help
print(table['Type'].value_counts())
from imblearn.under_sampling import ClusterCentroids
cc = ClusterCentroids(random_state = 22222)
x2, y2 = cc.fit_resample(table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],
                           table['Type'])
print(y2.value_counts())
```

Ln: 33 Col: 24

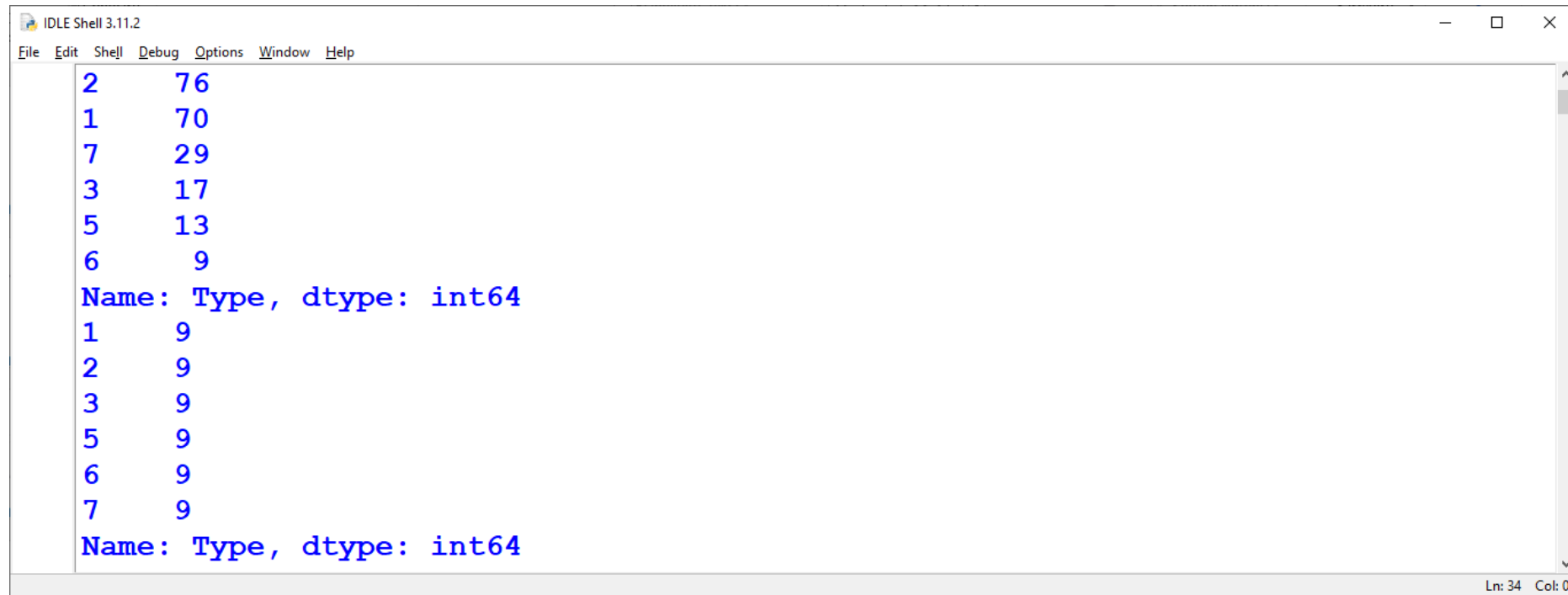
# Undersampling

---

```
print(table['Type'].value_counts())  
from imblearn.under_sampling import ClusterCentroids  
cc = ClusterCentroids(random_state = 22222)  
x2, y2 = cc.fit_resample(table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],  
                           table['Type'])  
print(y2.value_counts())
```

# Запуск

---



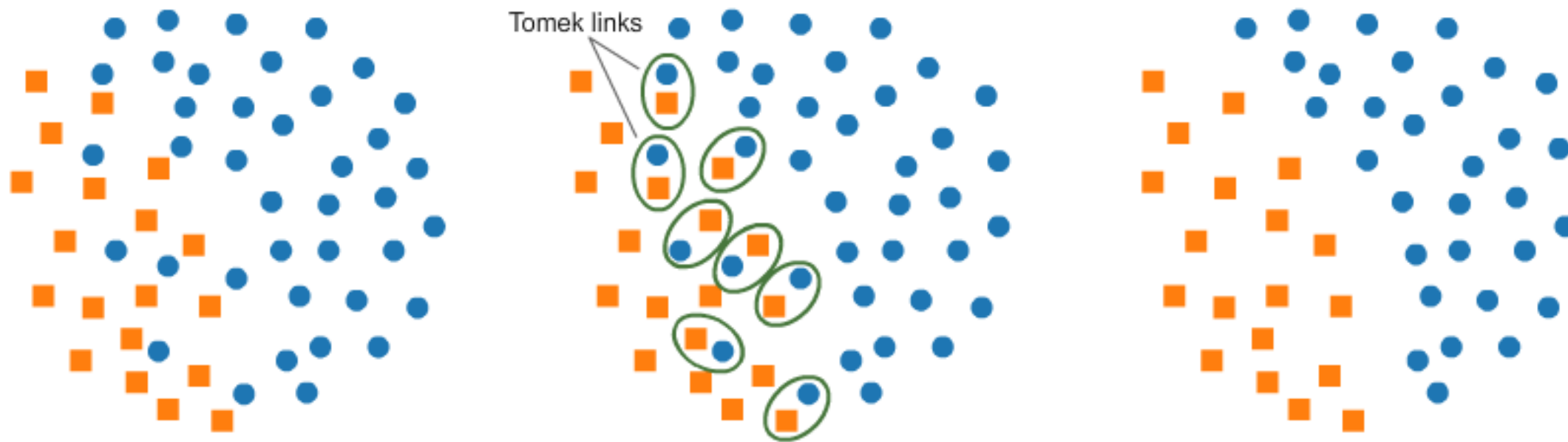
The screenshot shows a window titled "IDLE Shell 3.11.2" with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The main text area contains the following output:

```
2      76
1      70
7      29
3      17
5      13
6       9
Name: Type, dtype: int64
1       9
2       9
3       9
5       9
6       9
7       9
Name: Type, dtype: int64
```

The status bar at the bottom right indicates "Ln: 34 Col: 0".

# Пример сложной стратегии Undersampling Связи Томека (Tomek Links)

**Tomek Links** – это такой вид алгоритма Undersampling при котором удаляют не произвольные элементы большого класса, а те, которые «мешают» выявлять различия. То есть те, которые рядом с элементами другого класса.



Источник изображения – <https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>

# Tomek Links

---

```
example2-2runner.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecture8\example2-2runner.py (3.11.2)
File Edit Format Run Options Window Help

print(table['Type'].value_counts())
from imblearn.under_sampling import TomekLinks
tl = TomekLinks()
x3, y3 = tl.fit_resample(table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],
                             table['Type'])
print(y3.value_counts())

Ln: 24 Col: 0
```



# Tomek Links

---

```
print(table['Type'].value_counts())  
from imblearn.under_sampling import TomekLinks  
tl = TomekLinks()  
x3, y3 = tl.fit_resample(table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],  
                           table['Type'])  
print(y3.value_counts())
```

# Запуск

---



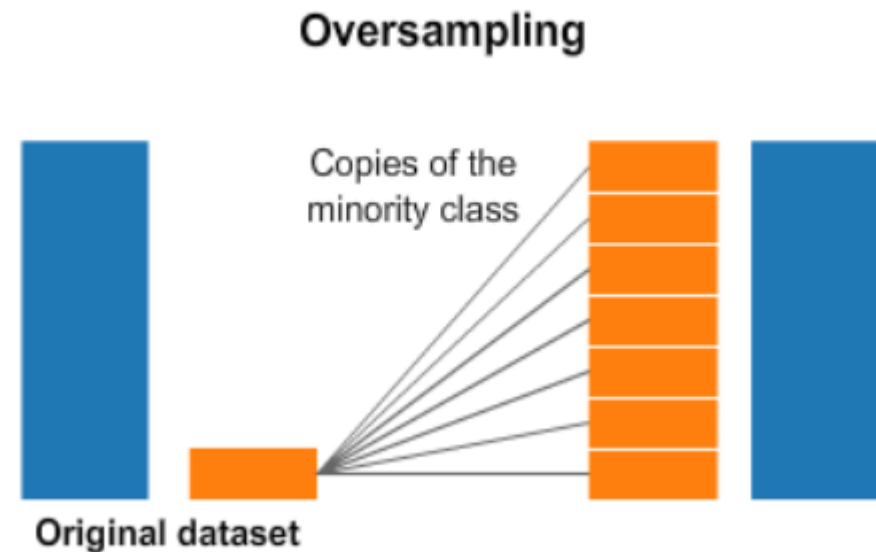
The screenshot shows the IDLE Shell 3.11.2 window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main text area displays two data series, each consisting of a list of numbers followed by a type declaration. The first series has values 2, 1, 7, 3, 5, 6 and a type of int64. The second series has values 2, 1, 7, 3, 5, 6 and a type of int64. The status bar at the bottom right indicates 'Ln: 43 Col: 0'.

```
IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
2      76
1      70
7      29
3      17
5      13
6      9
Name: Type, dtype: int64
2      68
1      64
7      28
3      13
5      13
6      9
Name: Type, dtype: int64
Ln: 43 Col: 0
```

# Стратегии балансировки выборки

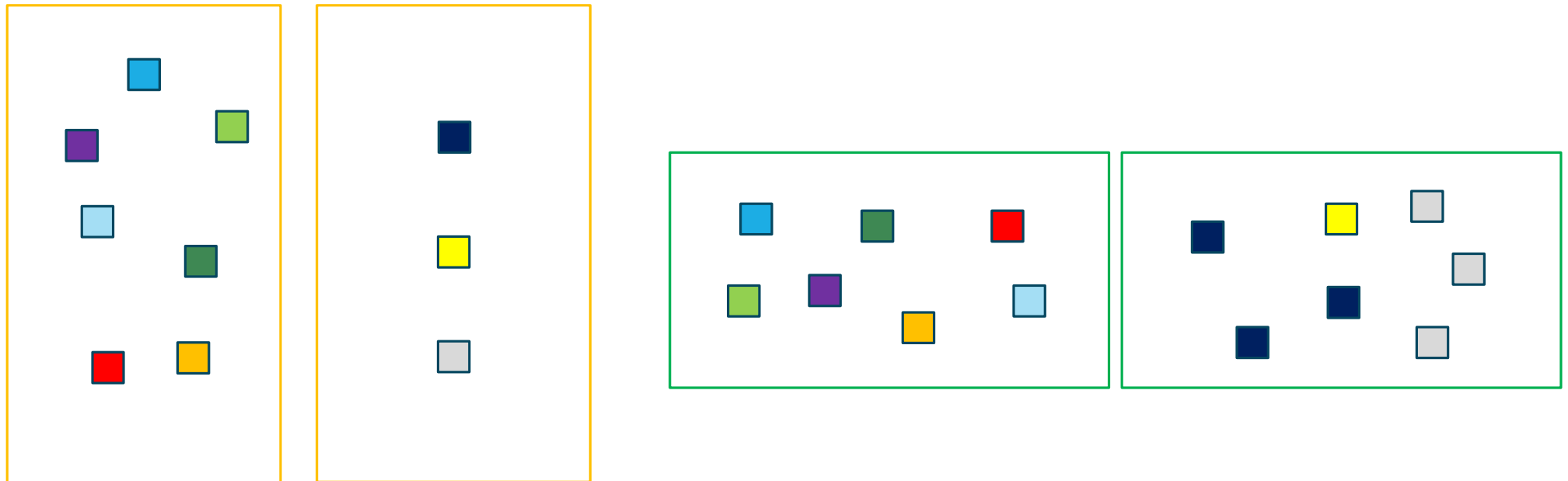
## Oversampling: копирование

**Oversampling** – это такой подход к формированию выборки, когда число строк всех классов доводят до числа строк самого большого из них. Проще всего это сделать дублированием, но могут быть и другие подходы.

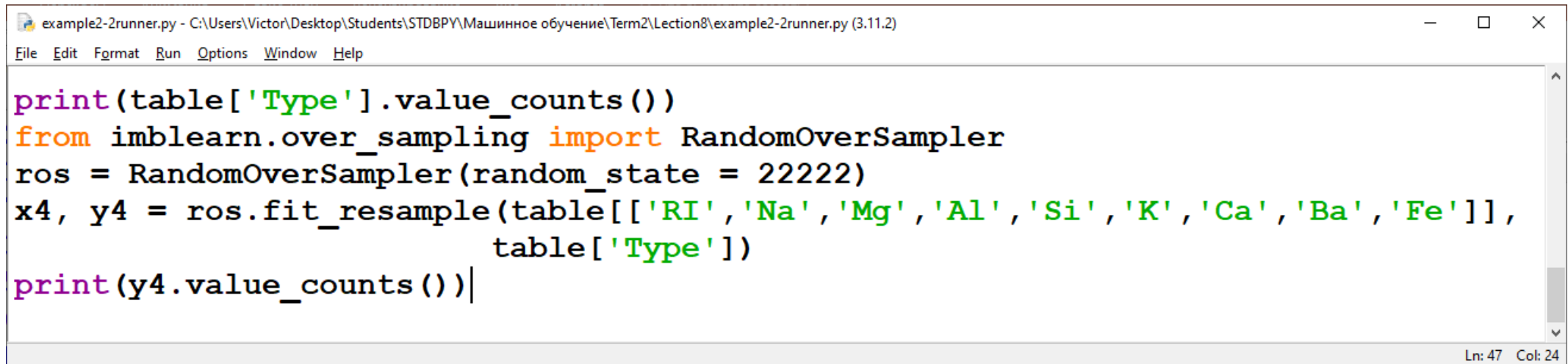


# Oversampling: случайный выбор с дублированием

**Случайный выбор с дублированием** – это такой подход к формированию выборки, когда для наименьшего класса выполняется случайное копирование элементов и их добавление в выборку до тех пор, пока нужный размер не будет достигнут.



# Случайный выбор с дублированием



```
example2-2runner.py - C:\Users\Victor\Desktop\Students\STDBPY\Машинное обучение\Term2\Lecion8\example2-2runner.py (3.11.2)
File Edit Format Run Options Window Help

print(table['Type'].value_counts())
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state = 22222)
x4, y4 = ros.fit_resample(table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],
                             table['Type'])
print(y4.value_counts())|
```


Ln: 47 Col: 24

# Случайный выбор с дублированием

---

```
print(table['Type'].value_counts())  
from imblearn.over_sampling import RandomOverSampler  
ros = RandomOverSampler(random_state = 22222)  
x4, y4 = ros.fit_resample(table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']],  
                           table['Type'])  
print(y4.value_counts())
```

# Запуск



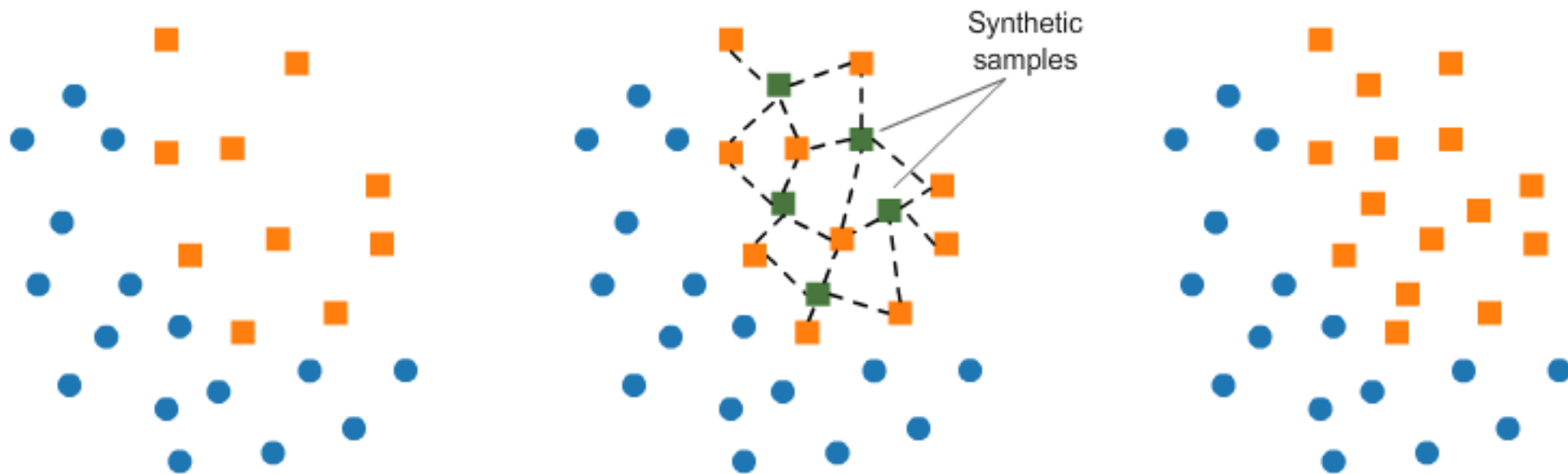
The screenshot shows the IDLE Shell 3.11.2 window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main text area displays the following output:

```
2      76
1      70
7      29
3      17
5      13
6       9
Name: Type, dtype: int64
1      76
2      76
3      76
5      76
6      76
7      76
Name: Type, dtype: int64
```

The status bar at the bottom right indicates "Ln: 151 Col: 0".

# Пример сложной стратегии Oversampling Синтетическая передискретизации (SMOTE)

**SMOTE** – это такой вид алгоритма Oversampling при котором создают новые элементы не путём копирования старых, а путем создания новых синтетических элементов, расположенных между несколькими реальными элементами миноритарного класса.

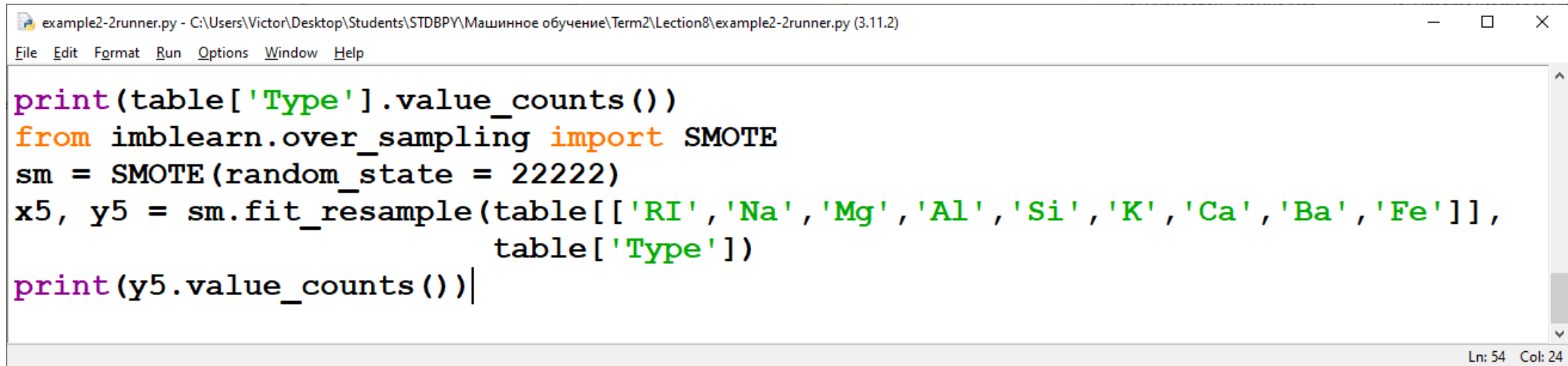


Источник изображения – <https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>



# SMOTE

---



```
example2-2runner.py - C:\Users\Victor\Desktop\Students\STDBPV\Машинное обучение\Term2\Lecture8\example2-2runner.py (3.11.2)
File Edit Format Run Options Window Help

print(table['Type'].value_counts())
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 22222)
x5, y5 = sm.fit_resample(table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']],
                             table['Type'])
print(y5.value_counts())|
```

Ln: 54 Col: 24

# SMOTE

---

```
print(table['Type'].value_counts())  
from imblearn.over_sampling import SMOTE  
sm = SMOTE(random_state = 22222)  
x5, y5 = sm.fit_resample(table[['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']],  
                           table['Type'])  
print(y5.value_counts())
```

# Запуск



The screenshot shows the IDLE Shell 3.11.2 window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main text area displays the following output:

```
2      76
1      70
7      29
3      17
5      13
6       9
Name: Type, dtype: int64
1      76
2      76
3      76
5      76
6      76
7      76
Name: Type, dtype: int64
```

The status bar at the bottom right indicates "Ln: 151 Col: 0".

# Формирование выборки

---

Может быть несколько ситуаций, когда использовать все имеющиеся данные нельзя.

Например, данных слишком много и алгоритм машинного обучения не сможет выполниться за разумное время. При этом большое число данных определенно не их разнообразием логическим, а просто большим числом повторений похожих ситуаций.

Некоторые алгоритмы требуют выделения обучающей, проверочной и тестовой выборок. Задача разделений на три (или две) выборки эквивалентна задаче выделения выборки меньшего размера.

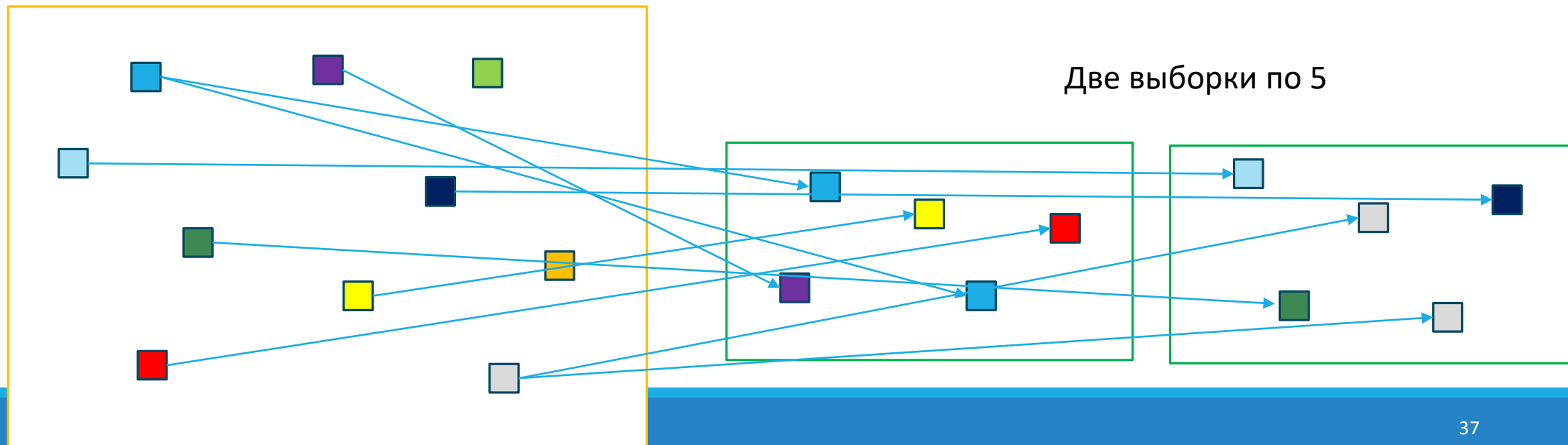
# Методы формирования выборки

## Простой случайный выбор с возвращением

У нас есть данные, объём которых  $N$  строк.

Задача получить выборку размеров в  $n$  строк ( $n < N$ ).

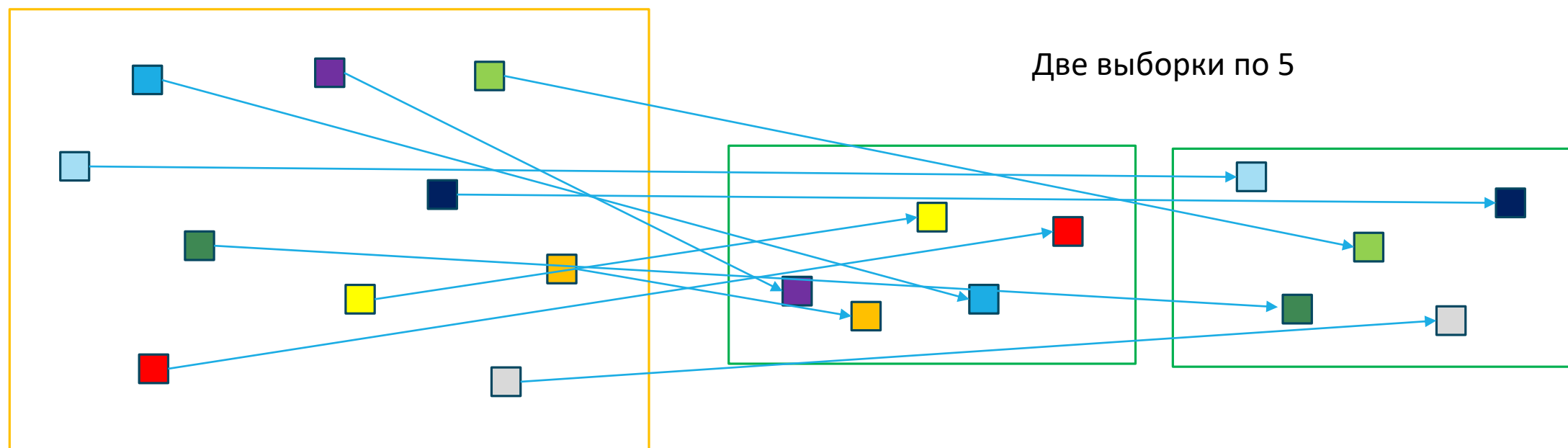
При простом случайном выборе с возвращением мы по очереди случайным образом берём элемент из исходных данных и копируем его значение в выборку. Сам элемент при этом из исходного набора не исчезает и впоследствии может попасть в выборку при следующих итерациях.



# Методы формирования выборки

## Простой случайный выбор без возвращения

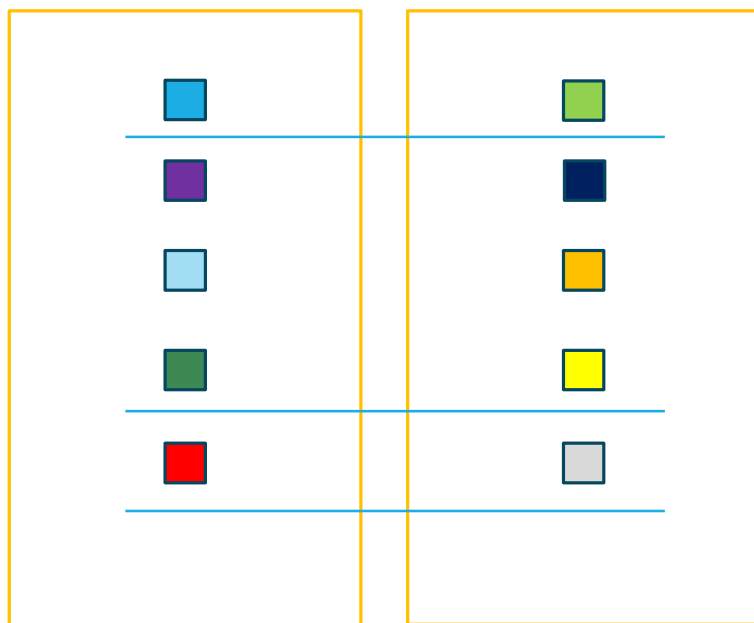
При простом случайном выборе без возвращения мы по одному случайным образом берём элемент из исходных данных и переносим его в выборку. Сам элемент при этом из исходного набора исчезает и впоследствии не может попасть в выборку при следующих итерациях повторно.



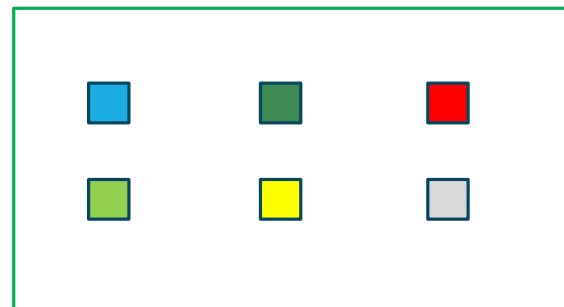
# Методы формирования выборки

## Систематическая случайная выборка

При систематической случайной выборке исходный набор данных разбивается на  $k = N/n$  кусков. Из первого куска элемент выбирается случайным образом. А из всех остальных кусков просто берутся элементы с такими же номерами.



1 выборка размером 6,  
случайно выпали индексы 1, 4, 5



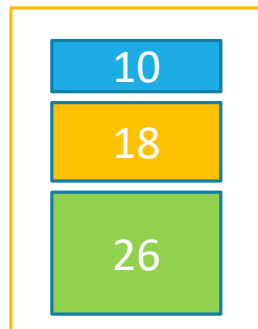
# Методы формирования выборки

## Стратифицированная выборка

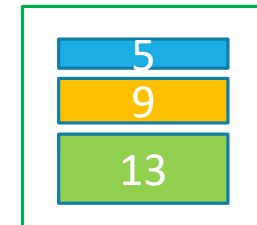
При стратифицированной выборке основная задача сохранить пропорцию с которой встречаются элементы разных классов. Для этого все данные делаются на отдельные наборы, в каждом из которых представители одного класса. Составляется пропорции на основе которой из каждого класса выбирается меньшее число элементов.

Например, пусть у нас в исходном наборе 30 записей. Из них 15 класса А, 5 класса В и 10 класса С. Пусть нам нужно собрать выборку в 12 элементов.

Тогда  $A/B = 3$ ,  $A/C = 3/2$ ,  $B/C = 1/2$ ,  $A/N = 1/2$ ,  $B/N = 1/6$ ,  $C/N = 1/3$ . Все эти пропорции должны сохраниться и в выборке. Таким образом, нам надо выбрать случайным образом 6 строк класса А, 2 строки класса В и 4 строки класса С.



Уменьшаем размер выборки в два раза



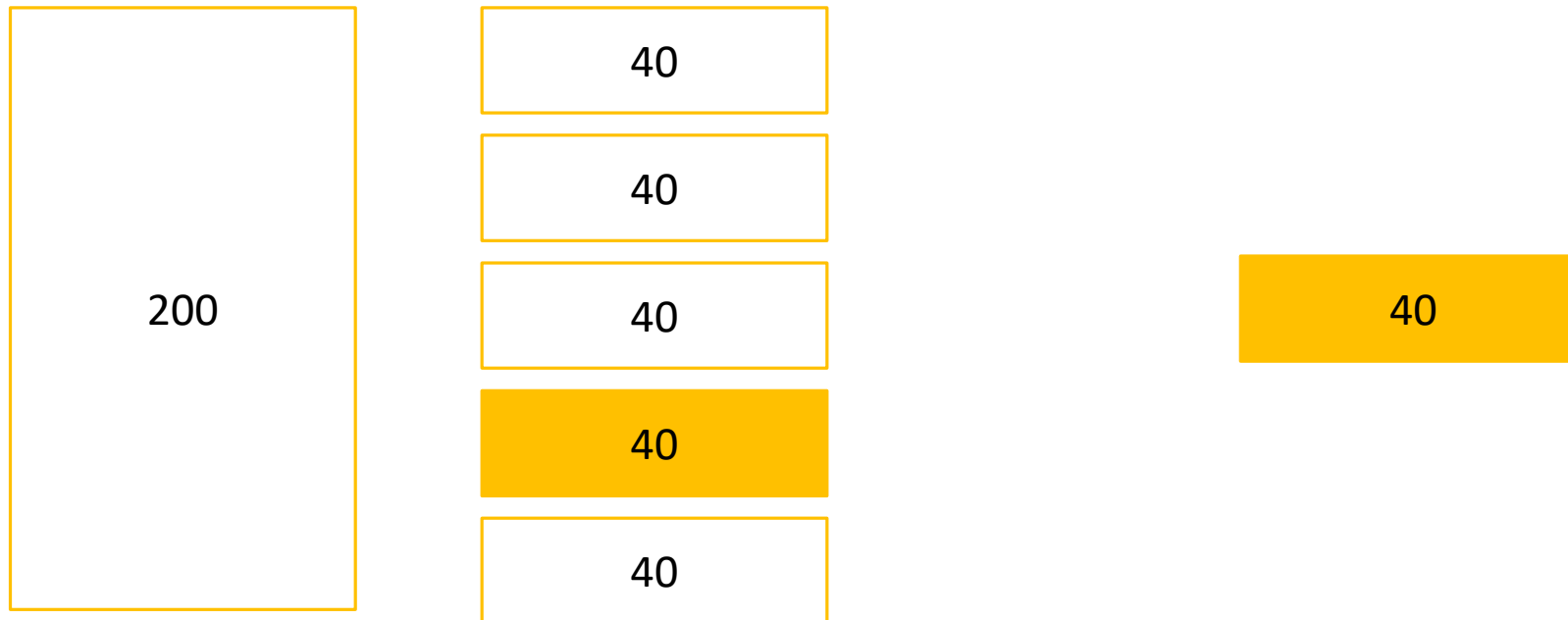


# Методы формирования выборки

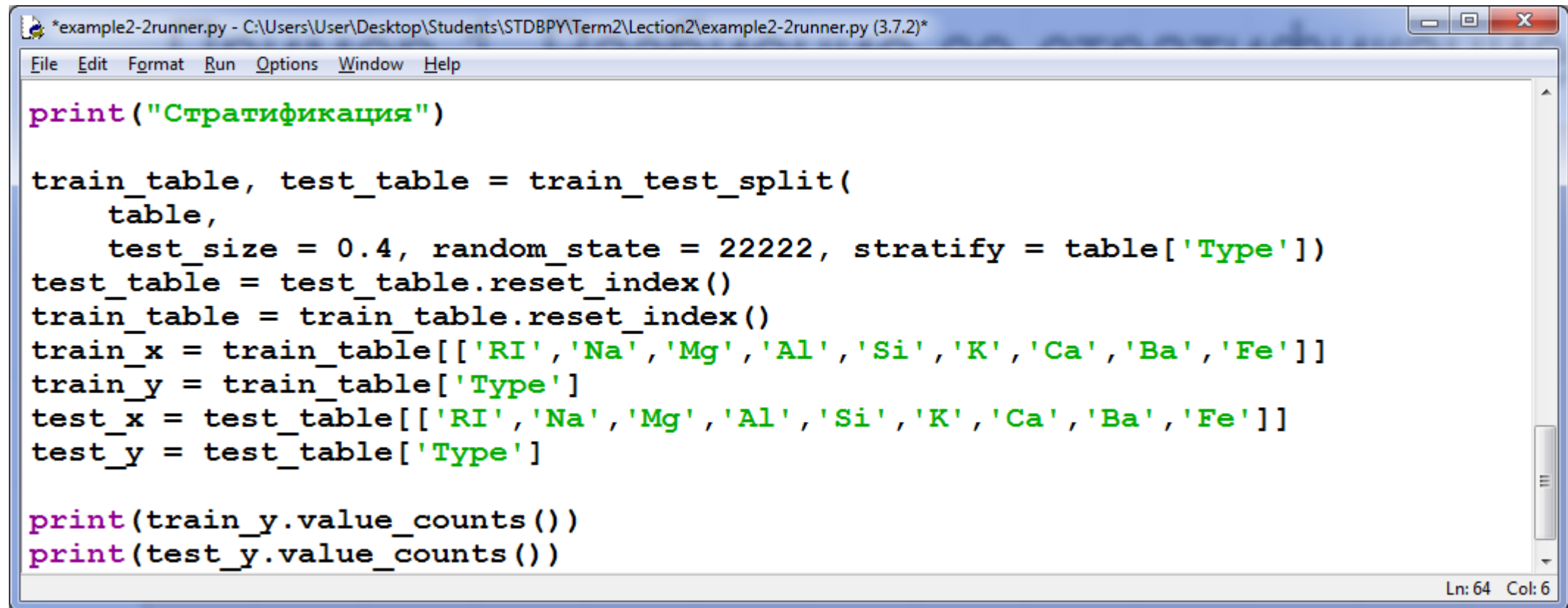
## Кластерная выборка

---

При кластерной выборке исходный набор данных нарезается на части, размером, требуемым для выборки. Из этих частей случайным образом выбирается одна. Такой подход самый быстрый.



# Пример 2. Разбиение со стратификацией



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

print("Стратификация")

train_table, test_table = train_test_split(
    table,
    test_size = 0.4, random_state = 22222, stratify = table['Type'])
test_table = test_table.reset_index()
train_table = train_table.reset_index()
train_x = train_table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
train_y = train_table['Type']
test_x = test_table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
test_y = test_table['Type']

print(train_y.value_counts())
print(test_y.value_counts())
```

Ln: 64 Col: 6

## Пример 2. Разбиение со стратификацией

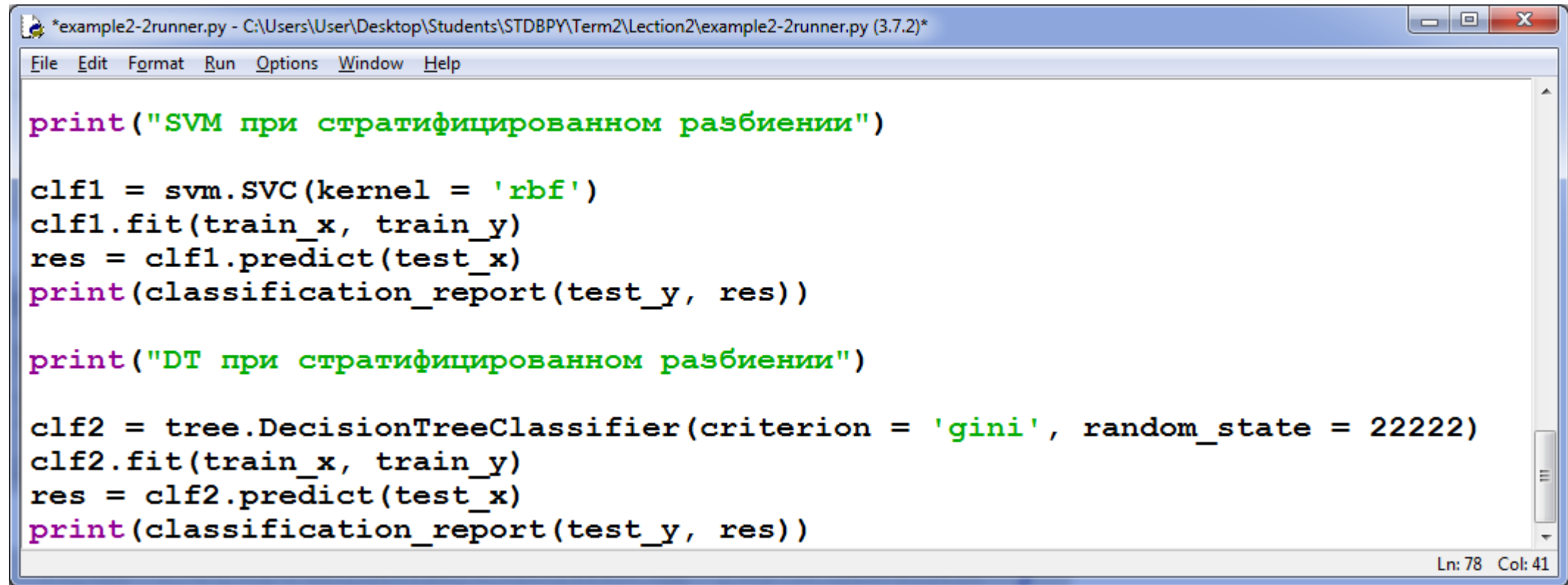
```
print("Стратификация")
train_table, test_table = train_test_split(table,
    test_size = 0.4, random_state = 22222, stratify = table['Type'])
test_table = test_table.reset_index()
train_table = train_table.reset_index()
train_x = train_table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']]
train_y = train_table['Type']
test_x = test_table[['Rl','Na','Mg','Al','Si','K','Ca','Ba','Fe']]
test_y = test_table['Type']
print(train_y.value_counts())
print(test_y.value_counts())
```

# Результат

---

```
Стратификация
2      46
1      42
7      17
3      10
5       8
6       5
Name: Type, dtype: int64
2      30
1      28
7      12
3       7
5       5
6       4
Name: Type, dtype: int64
```

# Пример 2. SVM и DT



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

print("SVM при стратифицированном разбиении")

clf1 = svm.SVC(kernel = 'rbf')
clf1.fit(train_x, train_y)
res = clf1.predict(test_x)
print(classification_report(test_y, res))

print("DT при стратифицированном разбиении")

clf2 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf2.fit(train_x, train_y)
res = clf2.predict(test_x)
print(classification_report(test_y, res))

Ln: 78 Col: 41
```

# Пример 2. SVM и DT

```
print("SVM при стратифицированном разбиении")
clf1 = svm.SVC(kernel = 'rbf')
clf1.fit(train_x, train_y)
res = clf1.predict(test_x)
print(classification_report(test_y, res))

print("DT при стратифицированном разбиении")
clf2 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf2.fit(train_x, train_y)
res = clf2.predict(test_x)
print(classification_report(test_y, res))
```

# Результат

---

SVM при стратифицированном разбиении

Warning (from warnings module):

File "C:\Users\User\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\metrics\\_classification.py", line 1268

\_warn\_prf(average, modifier, msg\_start, len(result))

UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

	precision	recall	f1-score	support
1	0.59	0.71	0.65	28
2	0.58	0.70	0.64	30
3	0.00	0.00	0.00	7
5	1.00	0.40	0.57	5
6	1.00	0.25	0.40	4
7	0.92	1.00	0.96	12
accuracy			0.65	86
macro avg	0.68	0.51	0.54	86
weighted avg	0.63	0.65	0.62	86

# Результат

---

DT при стратифицированном разбиении

	precision	recall	f1-score	support
1	0.56	0.50	0.53	28
2	0.55	0.60	0.57	30
3	0.20	0.29	0.24	7
5	0.50	0.20	0.29	5
6	0.00	0.00	0.00	4
7	0.80	1.00	0.89	12
accuracy			0.55	86
macro avg	0.43	0.43	0.42	86
weighted avg	0.53	0.55	0.53	86



# Сравним результаты\*

---

Метрика	SVM	SVM + Страт.	DT	DT + Страт.
Accuracy	0,65	0,65	0,71	0,55
Precision avg	0,53	0,68	0,53	0,43
Recall avg	0,42	0,51	0,56	0,43

Метод SVM при корректном разбиении только повысил качество, а вот для Decision Tree оказалось, что результат плачевен! Попробуем его улучшить, оставив стратифицированное разбиение.

\* – сравнение не совсем корректно, из-за разных тестовых выборок, но мы пренебрежём данным моментом. В идеале, нужна общая валидационная выборка.

# Идея 1

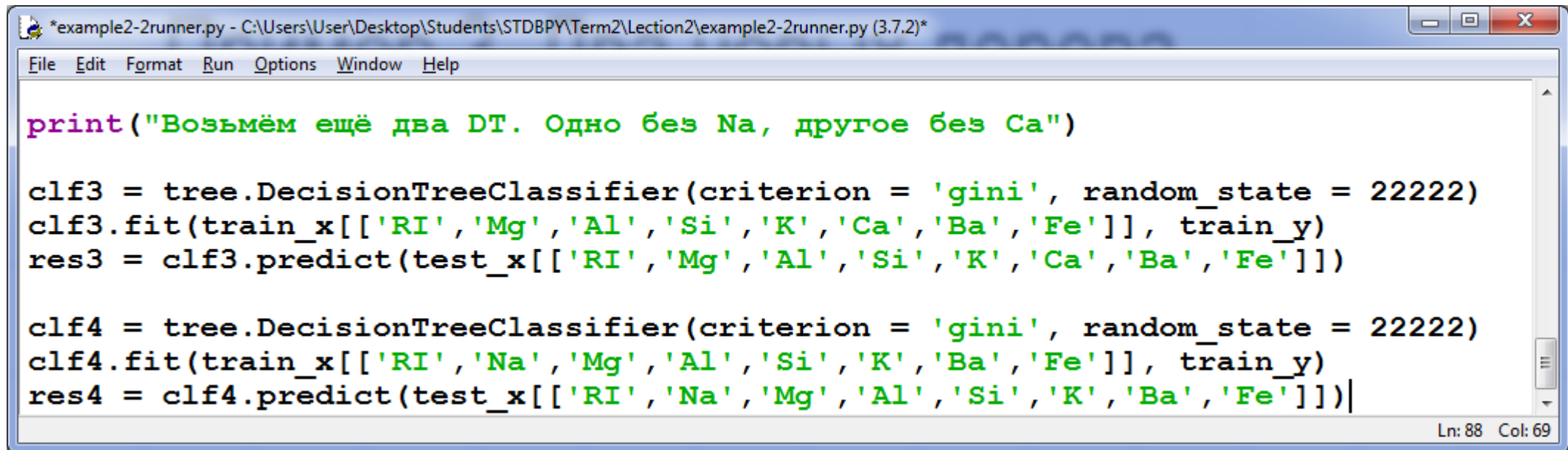
---

Если одно дерево справилось плохо, то может быть три дерева отработают лучше?

Данный подход называется ансамблевым. При таком подходе за счёт объединения нескольких «плохих» классификаторов получают новый «хороший». Различают идеи на основе бустинга (*boosting*) и бэггинга (*bagging*). Сегодня мы рассмотрим алгоритм на основе бэггинга Бреймана, но сначала попробуем что-то совсем простое.

Помимо полученного дерева построим ещё два. Одно будет при классификации не учитывать параметр  $N_a$ , другое не будет учитывать параметр  $C_a$ . Если оба этих дерева дадут одинаковый ответ и он не совпадёт с ответом первого дерева, то правильным ответом будем считать новый вариант. Иначе, правильный ответ старый.

# Пример 3. Два новых дерева



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

print("Возьмём ещё два DT. Одно без Na, другое без Ca")

clf3 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf3.fit(train_x[['RI', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']], train_y)
res3 = clf3.predict(test_x[['RI', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']])

clf4 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf4.fit(train_x[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ba', 'Fe']], train_y)
res4 = clf4.predict(test_x[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ba', 'Fe']])|
```

Ln: 88 Col: 69

# Пример 3. Два новых дерева

```
print("Возьмём ещё два DT. Одно без Na, другое без Ca")
```

```
clf3 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
```

```
clf3.fit(train_x[['Rl','Mg','Al','Si','K','Ca','Ba','Fe']], train_y)
```

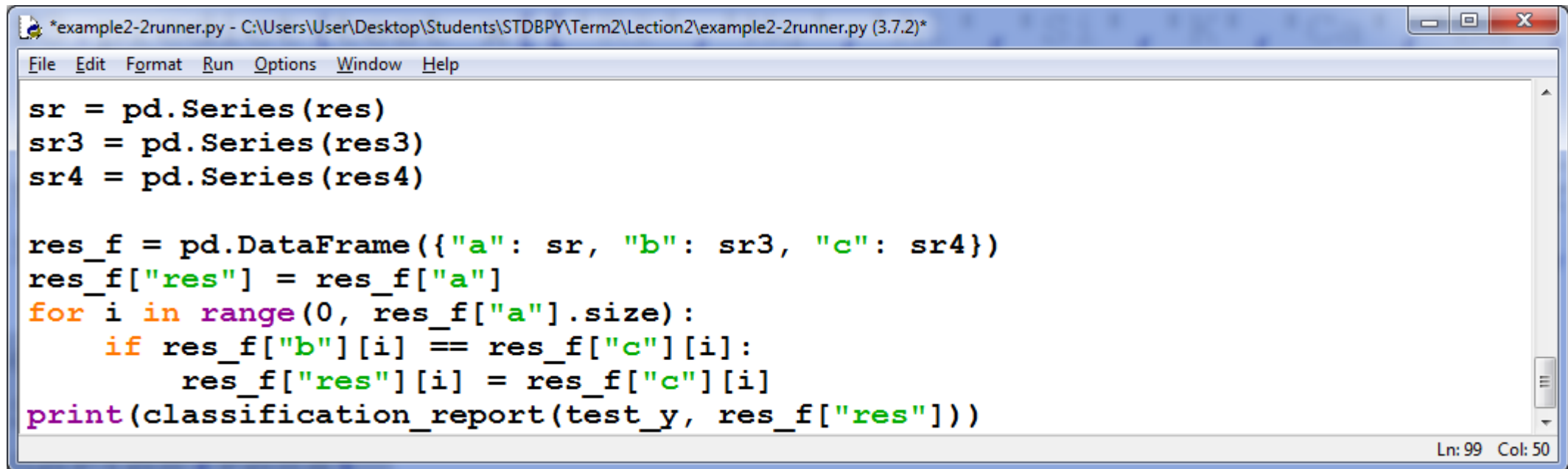
```
res3 = clf3.predict(test_x[['Rl','Mg','Al','Si','K','Ca','Ba','Fe']])
```

```
clf4 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
```

```
clf4.fit(train_x[['Rl','Na','Mg','Al','Si','K','Ba','Fe']], train_y)
```

```
res4 = clf4.predict(test_x[['Rl','Na','Mg','Al','Si','K','Ba','Fe']])
```

# Пример 3. Соберём новый результат



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

sr = pd.Series(res)
sr3 = pd.Series(res3)
sr4 = pd.Series(res4)

res_f = pd.DataFrame({"a": sr, "b": sr3, "c": sr4})
res_f["res"] = res_f["a"]
for i in range(0, res_f["a"].size):
    if res_f["b"][i] == res_f["c"][i]:
        res_f["res"][i] = res_f["c"][i]
print(classification_report(test_y, res_f["res"]))

Ln: 99 Col: 50
```

# Пример 3. Соберём новый результат

```
sr = pd.Series(res)
sr3 = pd.Series(res3)
sr4 = pd.Series(res4)

res_f = pd.DataFrame({"a": sr, "b": sr3, "c": sr4})
res_f["res"] = res_f["a"]
for i in range(0, res_f["a"].size):
    if res_f["b"][i] == res_f["c"][i]:
        res_f["res"][i] = res_f["c"][i]
print(classification_report(test_y, res_f["res"]))
```

# Результат

---

```
Возьмём ещё два DT. Одно без Na, другое без Ca
precision    recall  f1-score   support

     1      0.61      0.50      0.55        28
     2      0.60      0.60      0.60        30
     3      0.36      0.57      0.44         7
     5      0.67      0.40      0.50         5
     6      0.67      0.50      0.57         4
     7      0.75      1.00      0.86        12

 accuracy          0.60        86
 macro avg      0.61      0.60      0.59        86
weighted avg      0.61      0.60      0.60        86
```

# Сравним результаты

---

Метрика	SVM	SVM + Страт.	DT	DT + Страт.	3DT
Accuracy	0,65	0,65	0,71	0,55	0,60
Precision avg	0,53	0,68	0,53	0,43	0,61
Recall avg	0,42	0,51	0,56	0,43	0,60

Одно дерево нам удалось обогнать!



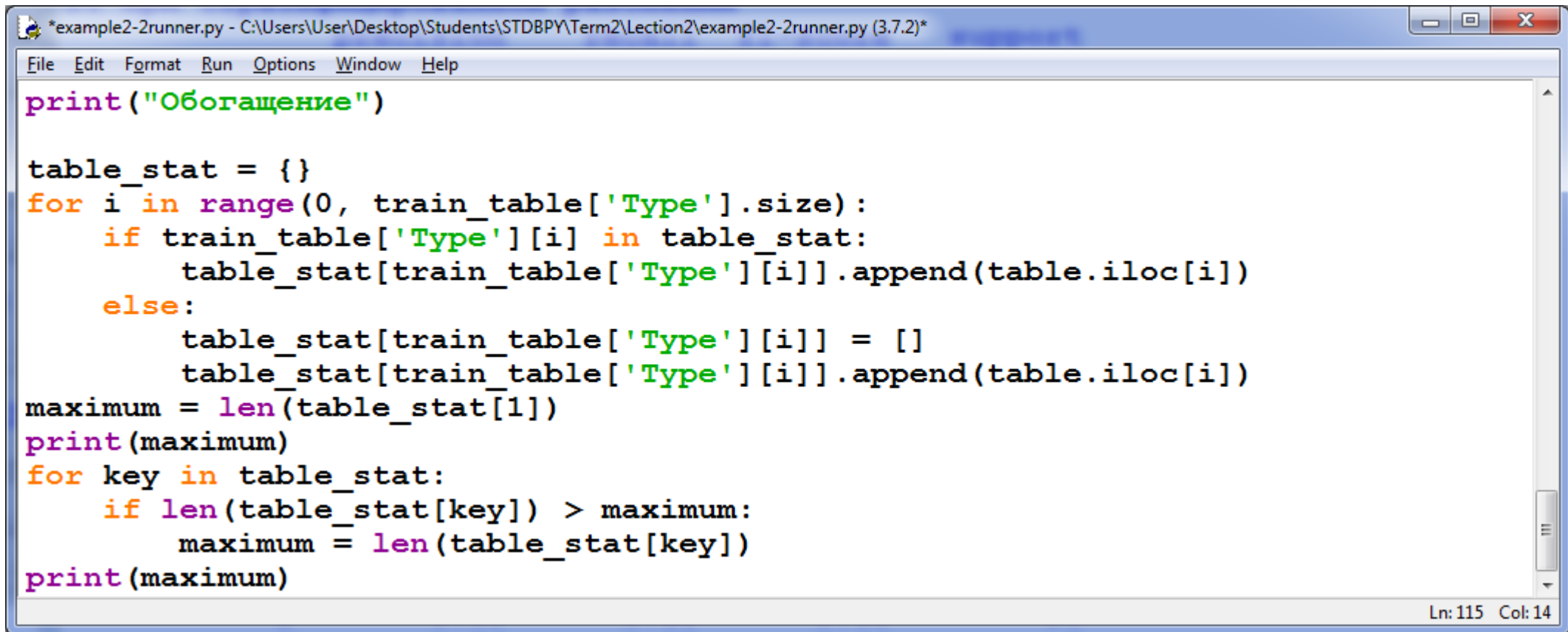
# Идея 2

---

После стратификации дерево решение выдало плохой результат. Отчасти это связано с тем, что в обучающей выборке мало представителей некоторых классов и их дискриминируют из-за количества. Попробуем решить проблему за счёт увеличения числа записей данных классов. Такой подход называется обогащением или Oversampling.

Мы используем самописный алгоритм, создающий копии существующих записей нужное число раз.

# Пример 4. Расчёт статистики



```
*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)*
File Edit Format Run Options Window Help

print("Обогащение")

table_stat = {}
for i in range(0, train_table['Type'].size):
    if train_table['Type'][i] in table_stat:
        table_stat[train_table['Type'][i]].append(table.iloc[i])
    else:
        table_stat[train_table['Type'][i]] = []
        table_stat[train_table['Type'][i]].append(table.iloc[i])
maximum = len(table_stat[1])
print(maximum)
for key in table_stat:
    if len(table_stat[key]) > maximum:
        maximum = len(table_stat[key])
print(maximum)
```

Ln: 115 Col: 14

# Пример 4. Расчёт статистики

```
print("Обогащение")

table_stat = {}

for i in range(0, train_table['Type'].size):

    if train_table['Type'][i] in table_stat:

        table_stat[train_table['Type'][i]].append(table.iloc[i])

    else:

        table_stat[train_table['Type'][i]] = []

        table_stat[train_table['Type'][i]].append(table.iloc[i])

maximum = len(table_stat[1])

print(maximum)

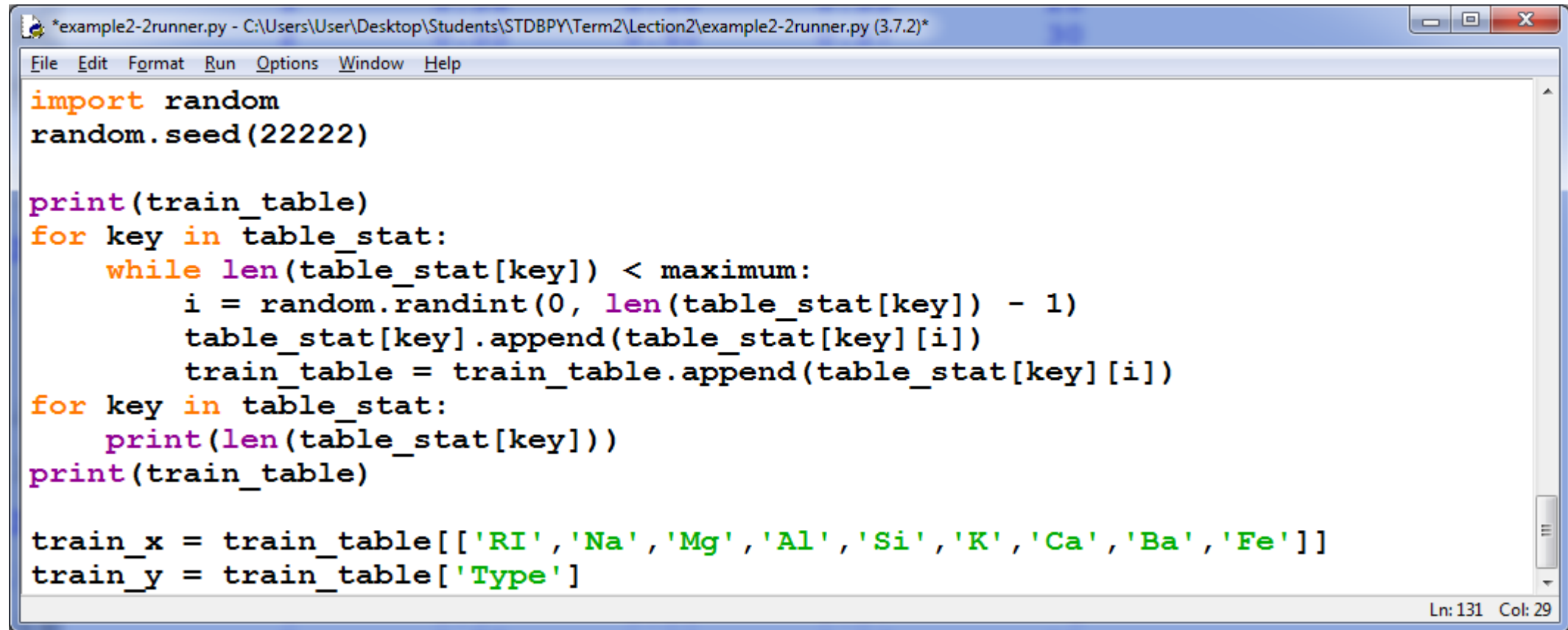
for key in table_stat:

    if len(table_stat[key]) > maximum:

        maximum = len(table_stat[key])

print(maximum)
```

# Пример 4. Обогащение

A screenshot of a Python IDE window titled '\*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)\*'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code is as follows:

```
import random
random.seed(22222)

print(train_table)
for key in table_stat:
    while len(table_stat[key]) < maximum:
        i = random.randint(0, len(table_stat[key]) - 1)
        table_stat[key].append(table_stat[key][i])
        train_table = train_table.append(table_stat[key][i])
for key in table_stat:
    print(len(table_stat[key]))
print(train_table)

train_x = train_table[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
train_y = train_table['Type']
```

The status bar at the bottom right shows 'Ln: 131 Col: 29'.

# Пример 4. Обогащение

```
import random

random.seed(22222)

print(train_table)

for key in table_stat:

    while len(table_stat[key]) < maximum:

        i = random.randint(0, len(table_stat[key]) - 1)

        table_stat[key].append(table_stat[key][i])

        train_table = train_table.append(table_stat[key][i])

for key in table_stat:

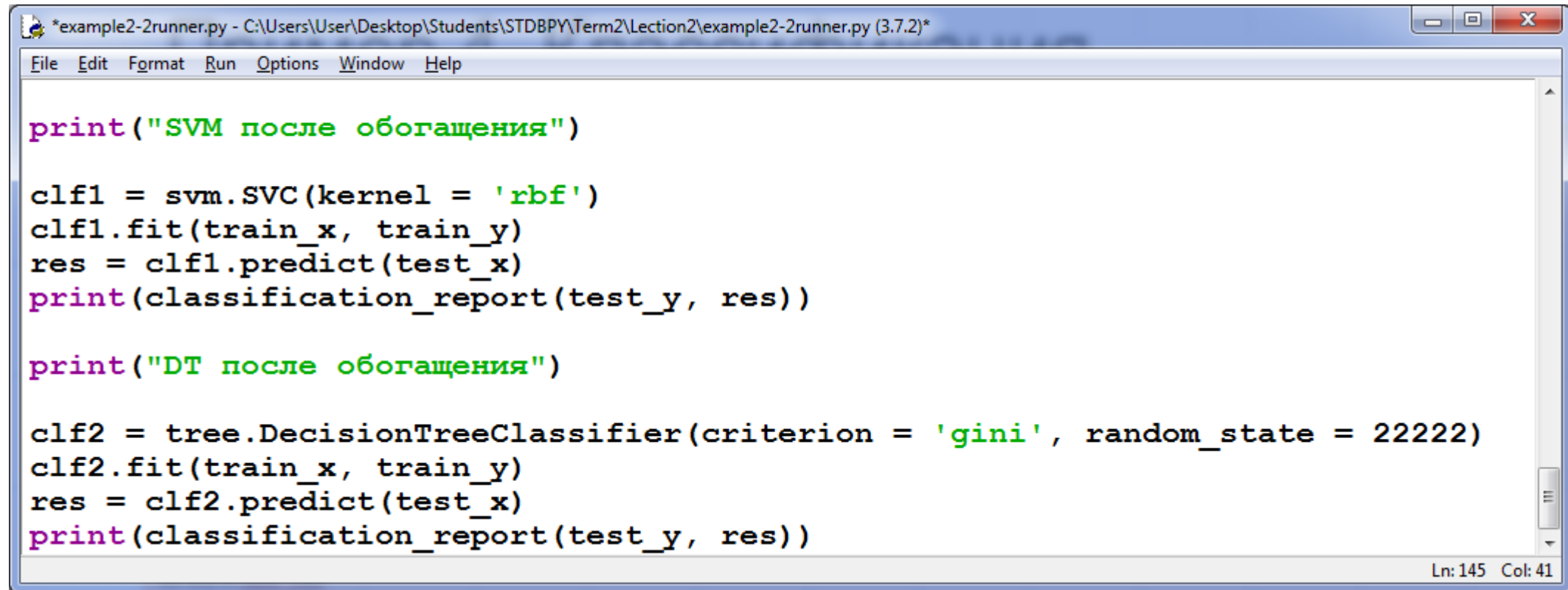
    print(len(table_stat[key]))

print(train_table)

train_x = train_table[['Rl', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

train_y = train_table['Type']
```

# Пример 4. Классификация



The image shows a screenshot of a Python IDE window titled '\*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lec2\example2-2runner.py (3.7.2)\*'. The window contains Python code for training and testing two classifiers: an SVM and a Decision Tree. The code is as follows:

```
print("SVM после обогащения")

clf1 = svm.SVC(kernel = 'rbf')
clf1.fit(train_x, train_y)
res = clf1.predict(test_x)
print(classification_report(test_y, res))

print("DT после обогащения")

clf2 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)
clf2.fit(train_x, train_y)
res = clf2.predict(test_x)
print(classification_report(test_y, res))
```

The status bar at the bottom right of the window indicates 'Ln: 145 Col: 41'.

# Пример 4. Классификация

```
print("SVM после обогащения")  
clf1 = svm.SVC(kernel = 'rbf')  
clf1.fit(train_x, train_y)  
res = clf1.predict(test_x)  
print(classification_report(test_y, res))  
print("DT после обогащения")  
clf2 = tree.DecisionTreeClassifier(criterion = 'gini', random_state = 22222)  
clf2.fit(train_x, train_y)  
res = clf2.predict(test_x)  
print(classification_report(test_y, res))
```

# Результат

SVM после обогащения

Warning (from warnings module):

File "C:\Users\User\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\metrics\\_classification.py", line 1268

\_warn\_prf(average, modifier, msg\_start, len(result))

UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

	precision	recall	f1-score	support
1	0.58	0.79	0.67	28
2	0.59	0.67	0.62	30
3	0.00	0.00	0.00	7
5	1.00	0.40	0.57	5
6	1.00	0.50	0.67	4
7	1.00	0.83	0.91	12
accuracy			0.65	86
macro avg	0.69	0.53	0.57	86
weighted avg	0.64	0.65	0.63	86



# Результат

---

DT после обогащения					
	precision	recall	f1-score	support	
1	0.73	0.68	0.70	28	
2	0.58	0.63	0.60	30	
3	0.20	0.14	0.17	7	
5	0.80	0.80	0.80	5	
6	1.00	0.75	0.86	4	
7	0.79	0.92	0.85	12	
accuracy			0.66	86	
macro avg	0.68	0.65	0.66	86	
weighted avg	0.66	0.66	0.66	86	

# Сравним результаты

---

Метрика	SVM	SVM + Страт.	SVM об.	DT	DT + Страт.	3DT	DT об.
Accuracy	0,65	0,65	0,65	0,71	0,55	0,60	0,66
Precision avg	0,53	0,68	0,69	0,53	0,43	0,61	0,68
Recall avg	0,42	0,51	0,53	0,56	0,43	0,60	0,65

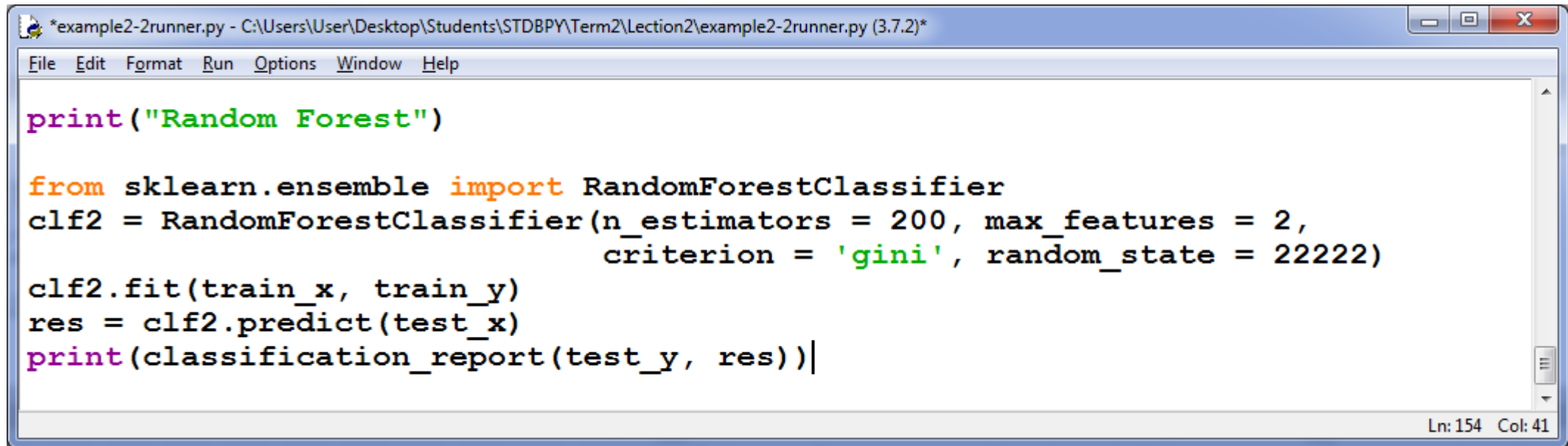
Опять лучше!

# Random Forest

---

Алгоритм Random Forest строит большое число деревьев полной глубины (если не задано ограничение) по различным кускам выборки и различному подмножеству параметров. Итоговый класс выбирается «голосованием».

# Пример 5. Random Forest

A screenshot of a Python IDE window titled '\*example2-2runner.py - C:\Users\User\Desktop\Students\STDBPY\Term2\Lecion2\example2-2runner.py (3.7.2)\*'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code is as follows:

```
print("Random Forest")

from sklearn.ensemble import RandomForestClassifier
clf2 = RandomForestClassifier(n_estimators = 200, max_features = 2,
                             criterion = 'gini', random_state = 22222)

clf2.fit(train_x, train_y)
res = clf2.predict(test_x)
print(classification_report(test_y, res))|
```

The status bar at the bottom right shows 'Ln: 154 Col: 41'.

# Пример 5. Random Forest

```
print("Random Forest")

from sklearn.ensemble import RandomForestClassifier

clf2 = RandomForestClassifier(n_estimators = 200, max_features = 2,
                             criterion = 'gini', random_state = 22222)

clf2.fit(train_x, train_y)

res = clf2.predict(test_x)

print(classification_report(test_y, res))
```

# Результат

---

## Random Forest

	precision	recall	f1-score	support
1	0.69	0.79	0.73	28
2	0.72	0.87	0.79	30
3	1.00	0.14	0.25	7
5	1.00	0.20	0.33	5
6	1.00	0.75	0.86	4
7	0.92	1.00	0.96	12
accuracy			0.76	86
macro avg	0.89	0.62	0.65	86
weighted avg	0.79	0.76	0.73	86

# Сравним результаты

---

Метрика	SVM	SVM + Страт.	SVM об.	DT	DT + Страт.	3DT	DT об.	RF
Accuracy	0,65	0,65	0,65	0,71	0,55	0,60	0,66	0,76
Precision avg	0,53	0,68	0,69	0,53	0,43	0,61	0,68	0,89
Recall avg	0,42	0,51	0,53	0,56	0,43	0,60	0,65	0,62

# Попробуем более сложные разбиения

---

Используем две новых стратегии:

- ❑ Tomek Links и стратификацию для обычного дерева решений и SVM;
- ❑ SMOTE для обычного дерева решений и SVM.

При этом Tomek Links мы применим к тому разбиению, которое у нас было при стратификации. Тестовая выборка не измениться. Поменяется только обучающая.

SMOTE мы применим к самому первому разбиению (без стратификации и обогащения). Тестовая выборка, опять, не измениться. Обогащаться только обучающая.



# Сравним результаты

---

Метрика	SVM	SVM + Страт.	SVM об.	DT	DT + Страт.	3DT	DT об.	RF	SVM + Страт + TL	DT + Страт + TL	SVM + SMOTE	DT + SMOTE
Accuracy	0,65	0,65	0,65	0,71	0,55	0,60	0,66	0,76	0,65	0,59	0,71	0,86
Precision avg	0,53	0,68	0,69	0,53	0,43	0,61	0,68	0,89	0,68	0,57	0,74	0,88
Recall avg	0,42	0,51	0,53	0,56	0,43	0,60	0,65	0,62	0,51	0,49	0,76	0,80

# Интернет ресурсы и литература

---

1. <https://scikit-learn.org/stable/modules/svm.html>
2. <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC>
3. [https://scikit-learn.org/stable/auto\\_examples/plot\\_roc\\_curve\\_visualization\\_api.html#sphx-glr-auto-examples-plot-roc-curve-visualization-api-py](https://scikit-learn.org/stable/auto_examples/plot_roc_curve_visualization_api.html#sphx-glr-auto-examples-plot-roc-curve-visualization-api-py)
4. [https://scikit-learn.org/stable/tutorial/statistical\\_inference/model\\_selection.html](https://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html)
5. [https://scikit-learn.org/0.22/modules/generated/sklearn.model\\_selection.cross\\_validate.html](https://scikit-learn.org/0.22/modules/generated/sklearn.model_selection.cross_validate.html)
6. <https://scikit-learn.org/0.20/modules/generated/sklearn.metrics.auc.html#sklearn.metrics.auc>
7. <https://habr.com/ru/company/ods/blog/328372/>
8. [https://github.com/esokolov/ml-course-msu/blob/master/ML15/lecture-notes/Sem05\\_metrics.pdf](https://github.com/esokolov/ml-course-msu/blob/master/ML15/lecture-notes/Sem05_metrics.pdf)
9. <https://imbalanced-learn.org/stable/>