



Машинное обучение

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН, Д.Ю. КУПРИЯНОВ, Т.А. МАНАЕНКОВА

ЛЕКЦИЯ 2

Часть 1

CSV. PANDAS. SERIES

CSV-формат

CSV-файл – это простейший язык разметки данных.

Основные составляющие:

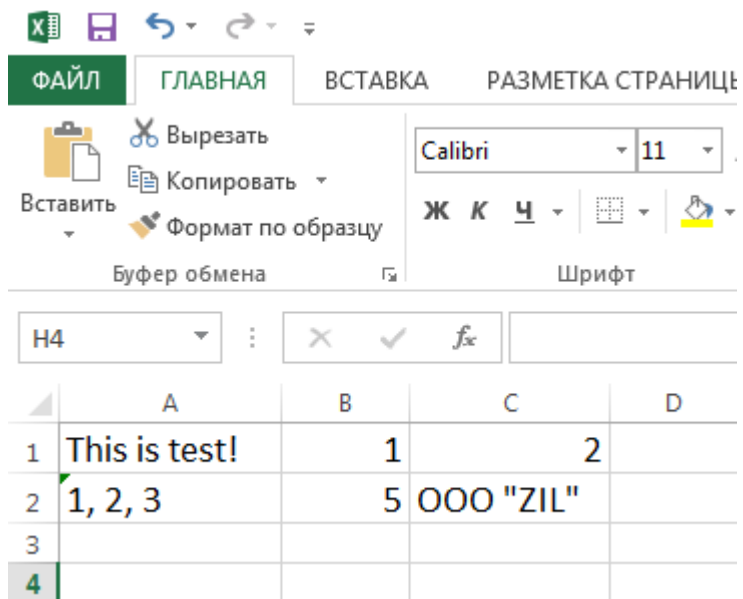
- разделитель строк;
- разделитель ячеек;
- ограничитель строк;
- данные.

Разделители и ограничители в CSV

Канонический формат CSV подразумевает, что каждая строка отделяется от другой символом конца строки и (но не обязательно) возврата каретки (либо \n, либо \n\r), каждая ячейка отделяется от другой символом запятой (,), а для сложных строковых значений (например, содержащих запятую) данные берутся внутрь двойных кавычек ("..."). При этом, если строка содержит ещё и сами кавычки, то содержимое экранируется заменой на две кавычки сразу: "ООО ""ЗИЛ""".

На сегодняшний день большинство систем вместо канонического формата CSV работают с форматом DSV (Delimiter-separated values). Данный формат позволяет использовать другие разделители ячеек и ограничители строк. Такой переход связан с различными факторами. Одним из них является необходимость учитывать локализацию. К примеру, в России целую часть числа от десятичной дроби принято отделять не точкой, а запятой, которая совпадает с разделителем по умолчанию. Поэтому Microsoft Excel с русской локализацией при сохранении данных в формате «CSV (разделитель – запятые)» выполняет сохранение с разделителем точка с запятой (;).

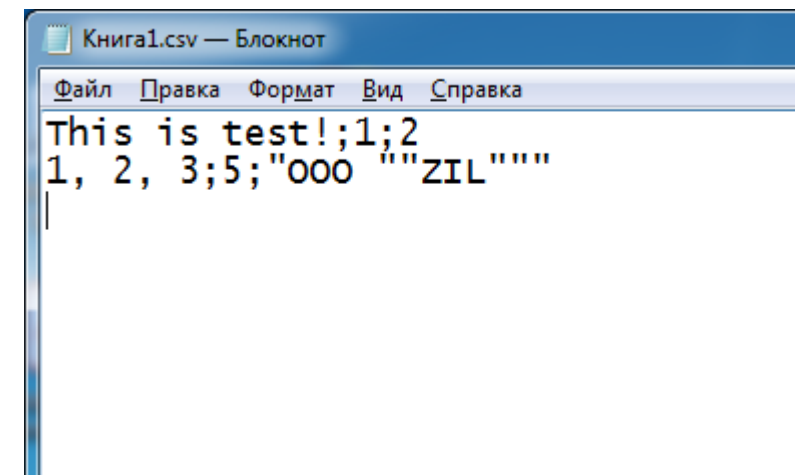
Пример



The screenshot shows the Microsoft Excel interface with the 'Главная' (Home) tab selected. The ribbon includes options for 'Файл', 'Главная', 'Вставка', and 'Разметка страницы'. The 'Буфер обмена' (Clipboard) group shows 'Вырезать' (Cut), 'Копировать' (Copy), and 'Вставить' (Paste) options. The 'Шрифт' (Font) group shows 'Calibri' font and '11' size. The spreadsheet area displays a table with 4 columns (A, B, C, D) and 4 rows. The data is as follows:

	A	B	C	D
1	This is test!	1	2	
2	1, 2, 3	5	000 "ZIL"	
3				
4				

Excel



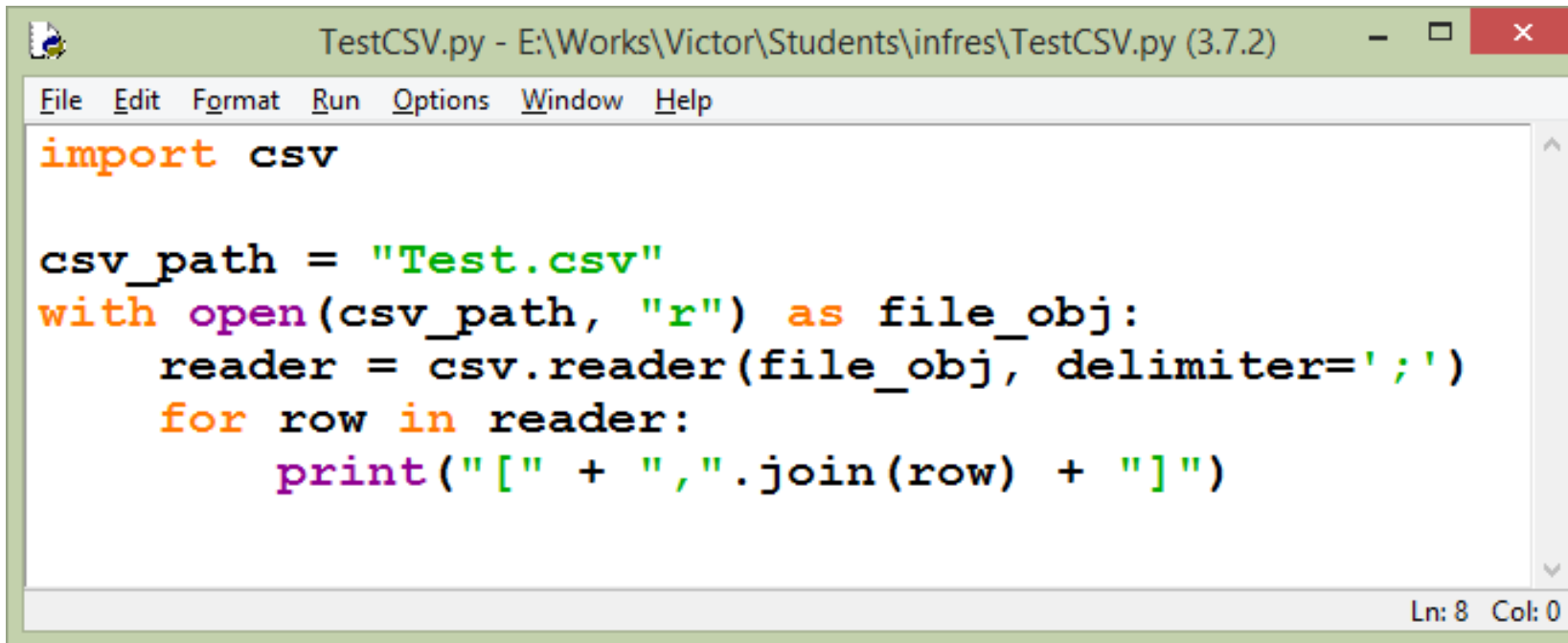
The screenshot shows a Notepad window titled 'Книга1.csv — Блокнот'. The menu bar includes 'Файл', 'Правка', 'Формат', 'Вид', and 'Справка'. The text content of the file is:

```
This is test!;1;2
1, 2, 3;5;"000 ""ZIL"""
```

Реальное содержимое CSV

Пример чтения и разбора CSV-файла

Библиотека csv встроена в стандартную поставку языка Python и позволяет выполнять как чтение и разбор файлов в данном формате, так и их запись.

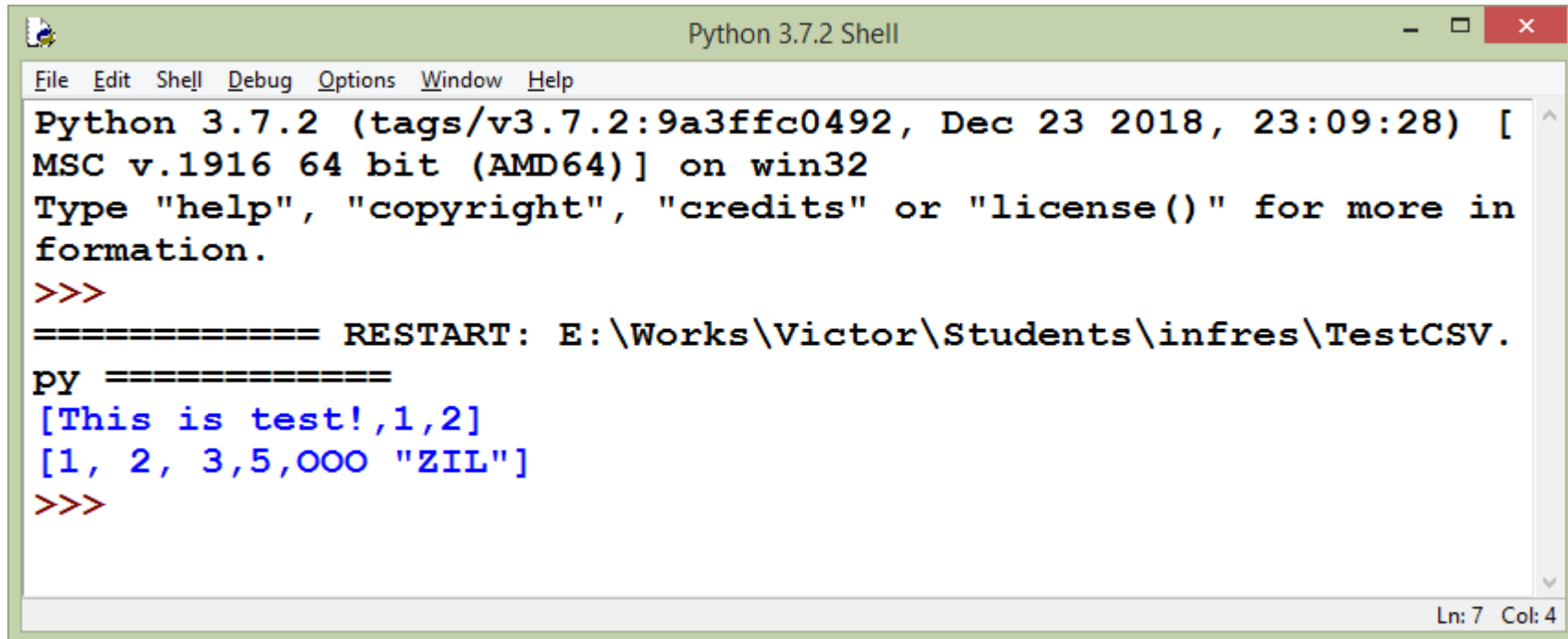


```
TestCSV.py - E:\Works\Victor\Students\infres\TestCSV.py (3.7.2)
File Edit Format Run Options Window Help
import csv

csv_path = "Test.csv"
with open(csv_path, "r") as file_obj:
    reader = csv.reader(file_obj, delimiter=';')
    for row in reader:
        print "[" + ",".join(row) + "]"

Ln: 8 Col: 0
```

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 in
formation.
>>>
===== RESTART: E:\Works\Victor\Students\infres\TestCSV.
py =====
[This is test!,1,2]
[1, 2, 3,5,000 "ZIL"]
>>>
```

Ln: 7 Col: 4

Лучше использовать Pandas!

Pandas (название происходит не от наименования животного, а является сокращением слов «panel data» – табличные данные) создавалась прежде всего для обеспечения быстрой и удобной работы с более сложными структурами данных, чем просто многомерные массивы.

За много лет развития библиотеки Pandas в неё были добавлены многочисленные иные возможности и в настоящее время она рассматривается, прежде всего, как инструмент для работы с «большими данными» (BigData) и как основа для «машинного обучения» (Machine Learning).

Pandas доступна по адресу [\[4\]](#).

Для установки pandas можно использовать команду*:

python.exe -m pip install pandas

* Здесь и далее приводятся команды для ОС Windows. Для ОС Linux команды будут без префикса python.exe -m.

Основные рабочие единицы Pandas

Основные рабочие единицы в Pandas – это **Series** и **DataFrame**.

DataFrame в библиотеке Pandas чем-то похож на обычные таблицы, которые мы видели в дисциплине «Базы данных и экспертные системы».

Series или по-русски ряды – это, с одной стороны, маркированный список (аналог dict). А с другой стороны – это аналог колонки таблицы.

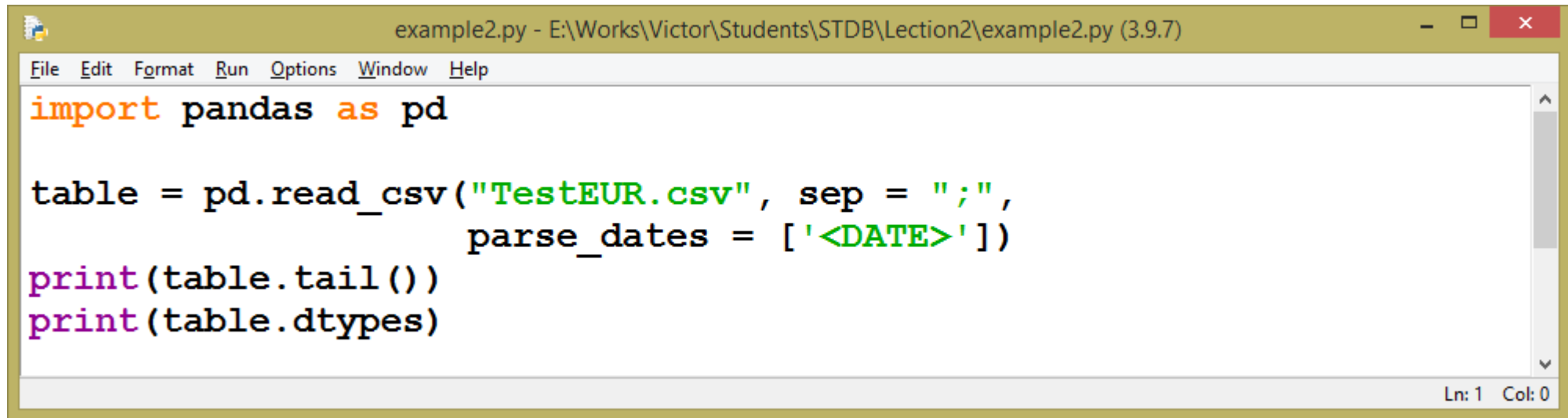
Загрузка из CSV с помощью Pandas

Библиотека Pandas обеспечивает загрузку данных из CSV-формата с помощью метода `read_csv`.

В качестве примера загрузим файл с курсом Евро с сайта <https://www.finam.ru/profile/forex/eur-rub/export>.

	A	B	C	D	E	F	G	H	I	J	K
1	<TICKER>	<PER>	<DATE>	<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>	<VOL>		
2	EURRUB	D	20181015	0	76.200000	76.586000	75.686000	75.963000	5371		
3	EURRUB	D	20181016	0	76.034000	76.078000	75.514000	75.573000	5486		
4	EURRUB	D	20181017	0	75.653000	75.871200	75.192000	75.272000	6831		
5	EURRUB	D	20181018	0	75.355000	75.763200	75.223000	75.263000	6902		
6	EURRUB	D	20181019	0	75.354000	75.479000	75.017000	75.226900	6885		
7	EURRUB	D	20181020	0	75.291000	75.392000	75.291000	75.392000	8		
8	EURRUB	D	20181021	0	75.179500	75.552000	75.179500	75.531300	54		
9	EURRUB	D	20181022	0	75.545000	75.682000	74.475000	74.665000	6438		
10	EURRUB	D	20181023	0	74.751000	75.453000	74.587000	75.034800	6859		
11	EURRUB	D	20181024	0	75.060000	75.377000	73.990000	74.755400	6974		
12	EURRUB	D	20181025	0	74.787000	75.146800	74.307000	74.531000	6933		
13	EURRUB	D	20181026	0	74.614000	75.073100	74.312000	74.678400	6978		

Пример



The screenshot shows a Python IDE window titled "example2.py - E:\Works\Victor\Students\STDB\Lection2\example2.py (3.9.7)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains the following Python code:

```
import pandas as pd

table = pd.read_csv("TestEUR.csv", sep = ";",
                    parse_dates = ['<DATE>'])
print(table.tail())
print(table.dtypes)
```

The status bar at the bottom right indicates "Ln: 1 Col: 0".

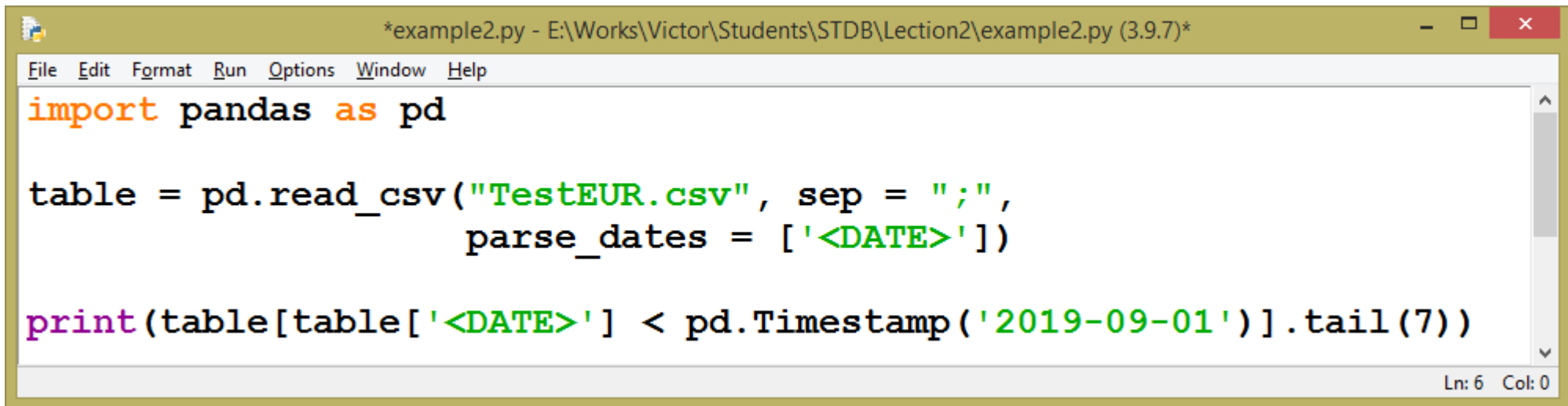
Результат

```
IDLE Shell 3.9.7
File Edit Shell Debug Options Window Help
>>>
===== RESTART: E:\Works\Victor\Students\STDB\Lec2\example2.py =====
  <TICKER> <PER>      <DATE>  <TIME>  <OPEN>  <HIGH>  <LOW>  <CLOSE>  <VOL>
349  EURRUB      D 2019-10-11      0  70.858  71.0210  70.575  70.8970  144237
350  EURRUB      D 2019-10-13      0  70.857  70.9180  70.675  70.7690   1942
351  EURRUB      D 2019-10-14      0  70.826  71.0450  70.576  70.8080  119901
352  EURRUB      D 2019-10-15      0  70.873  71.0590  70.659  70.9269  130350
353  EURRUB      D 2019-10-16      0  70.944  71.1835  70.717  70.9180  132236
<TICKER>          object
<PER>             object
<DATE>            datetime64[ns]
<TIME>            int64
<OPEN>            float64
<HIGH>            float64
<LOW>             float64
<CLOSE>           float64
<VOL>             int64
dtype: object
>>>
```

Ln: 21 Col: 4

Задание условий в []

DataFrame позволяет задать условия отсеивания в []. Например:

A screenshot of a Python IDE window titled '*example2.py - E:\Works\Victor\Students\STDB\Lecture2\example2.py (3.9.7)*'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor contains the following Python code:

```
import pandas as pd

table = pd.read_csv("TestEUR.csv", sep = ";",
                    parse_dates = ['<DATE>'])

print(table[table['<DATE>'] < pd.Timestamp('2019-09-01')].tail(7))
```

The status bar at the bottom right indicates 'Ln: 6 Col: 0'.

Результат

```

IDLE Shell 3.9.7
File Edit Shell Debug Options Window Help
>>>
===== RESTART: E:\Works\Victor\Students\STDB\Lec2\example2.py =====
  <TICKER> <PER>      <DATE>  <TIME>  <OPEN>  <HIGH>  <LOW>  <CLOSE>  <VOL>
307  EURRUB      D 2019-08-23      0  72.641  74.2000  72.404  74.2000  5579
308  EURRUB      D 2019-08-25      0  73.641  74.0080  73.516  73.7400   748
309  EURRUB      D 2019-08-26      0  73.855  73.8710  73.114  73.3873  6576
310  EURRUB      D 2019-08-27      0  73.405  73.9810  73.140  73.6110  6491
311  EURRUB      D 2019-08-28      0  73.686  74.2585  73.528  73.9997  6377
312  EURRUB      D 2019-08-29      0  74.011  74.0930  73.340  73.5320  6301
313  EURRUB      D 2019-08-30      0  73.546  73.6070  73.122  73.4230  5994
>>> |
Ln: 13 Col: 4

```

Series

По умолчанию каждый элемент ряда маркируется целым числом, начиная с 0.

► In [3]:

```
1 import pandas as pd
2
3 fruits = ["Яблоки сезонные", "Груши «Конференция»",
4           "Бананы", "Мандарины (Турция)", "Апельсины"]
5 prices = [99, 145, 75, 90, 125]
6 prices_series = pd.Series(prices)
7 fruits_series = pd.Series(fruits)
8 print(prices_series)
9 print(fruits_series)
```

```
0      99
1     145
2      75
3      90
4     125
dtype: int64
0      Яблоки сезонные
1  Груши «Конференция»
2           Бананы
3  Мандарины (Турция)
4           Апельсины
dtype: object
```

Series

Можно сделать
маркировку ряда в виде
объектов, например, строк.

► In [4]:

```
1 import pandas as pd
2
3 fruits = ["Яблоки сезонные", "Груши «Конференция»",
4           "Бананы", "Мандарины (Турция)", "Апельсины"]
5 prices = [99, 145, 75, 90, 125]
6 fruits_series = pd.Series(prices, index = fruits)
7 print(fruits_series)
```

```
Яблоки сезонные      99
Груши «Конференция»  145
Бананы                75
Мандарины (Турция)   90
Апельсины            125
dtype: int64
```


Типы данных Series

Хотя в Series можно загрузить данные разных типов, тем не менее, Pandas выберет один основной тип (если Вы его не указали явно), который будет использоваться при отображении и основных операциях.

Основные типы:

- float,
- int,
- bool,
- object,
- datetime64[ns],
- другие.

Пример

```
► In [11]: 1 import pandas as pd
           2
           3 numbers = [99, 145, 75, 90, 10.55]
           4 datas = [1, 2, 4.5, "This is test!"]
           5 numbers_series = pd.Series(numbers)
           6 datas_series = pd.Series(datas)
           7 print(numbers_series, datas_series)

0    99.00
1   145.00
2    75.00
3    90.00
4    10.55
dtype: float64 0          1
1              2
2              4.5
3    This is test!
dtype: object
```

Сложности двойной индексации: loc, iloc

Создадим простую серию с числами и возьмем срез:

```
import pandas as pd
num=pd.Series(range(0,1000,100))
sl=num[2:7]
print(sl)
```

```
2    200
3    300
4    400
5    500
6    600
dtype: int64
```

Индексы-метки остались закрепленными за своими значениями элементов.

```
sl[2]
```

```
200
```

Индекс-метка

```
sl[2:5]
```

```
0
```

```
4    400
```

```
5    500
```

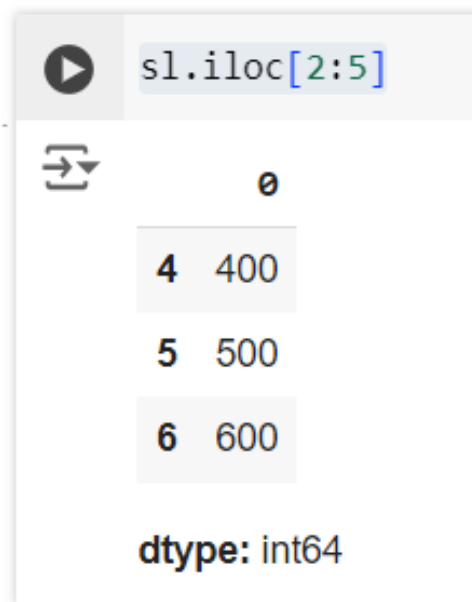
```
6    600
```

dtype: int64

Срез осуществляется по порядковым индексам

Сложности двойной индексации: loc, iloc

Достоверно достучаться к элементам Series по порядковому индексу можно через **локатор iloc**:



```
sl.iloc[2:5]
```

	0
4	400
5	500
6	600

dtype: int64

А достоверный доступ и срез через индексы-метки через **локатор loc**:



```
[8] sl.loc[2:5]
```

	0
2	200
3	300
4	400
5	500

dtype: int64

Обратите внимание, что элемент с последним индексом включен!
Поскольку метки являются ключами доступа.

Добавление/удаление данных

```
[10] s=pd.Series(data=[0,10,20,30] , index=[1,2,3,4])  
      print(s)
```

```
⇒ 1    0  
   2   10  
   3   20  
   4   30  
   dtype: int64
```

Добавление элемента с присваиванием индекса-метки через loc:

```
▶ s.pop(1)
```

```
⇒ 0
```

Удаление по
индекс-метке

```
[12] s.loc[5]=50  
      print(s)
```

```
⇒ 2    10  
   3    20  
   4    30  
   5    50  
   dtype: int64
```

Добавление/удаление данных

Но можно нарушить порядок:

```
▶ s.loc[1]=100  
print(s)
```

```
↕  
2    10  
3    20  
4    30  
5    50  
1    100  
dtype: int64
```

Тогда можно пересортировать индексы:

```
▶ s.sort_index(inplace=True)  
print(s)
```

```
↕  
1    100  
2     10  
3     20  
4     30  
5     50  
dtype: int64
```

Конкатенация

```
▶ a=pd.Series(data=[10,20,34] , index=[1,2,4])  
b=pd.Series(data=[100,200,300] , index=[1,2,3])  
c=pd.concat([a,b])  
print(c)
```

```
⇒ 1    10  
   2    20  
   4    34  
   1   100  
   2   200  
   3   300  
dtype: int64
```

```
▶ c[1]=1000  
print(c)
```

```
⇒ 1    1000  
   2     20  
   4     34  
   1    1000  
   2     200  
   3     300  
dtype: int64
```

Индексы-метки могут оказаться неуникальными!

Присваивание в такую метку приводит к изменению всех значений с этой меткой

```
▶ c.index.is_unique
```

```
⇒ False
```

Сброс индексов

```
▶ c[1]=1000  
print(c)
```

```
⇒ 1    1000  
   2      20  
   4      34  
   1    1000  
   2     200  
   3     300  
dtype: int64
```

```
▶ c.reset_index()
```

```
⇒
```

	index	0
0	1	1000
1	2	20
2	4	34
3	1	1000
4	2	200
5	3	300

Это уже DataFrame

```
▶ c.reset_index()[0]
```

```
⇒
```

	0
0	1000
1	20
2	34
3	1000
4	200
5	300

dtype: int64

Агрегирующие методы

```
import numpy as np
v=[1,2,3,np.nan,4,np.nan,3]
arr=np.array(v)
ser=pd.Series(v)
print(arr)
print(ser)
for i in arr, ser:
    print(i.min(),i.min(),i.sum())
```

```
[ 1.  2.  3. nan  4. nan  3.]
0    1.0
1    2.0
2    3.0
3    NaN
4    4.0
5    NaN
6    3.0
dtype: float64
nan nan nan
1.0 1.0 13.0
```

Агрегирующие методы по сравнению с массивами в сериях переопределены: в них допускаются (игнорируются) пустые ячейки

```
ser.value_counts()
```

```
count
3.0    2
1.0    1
2.0    1
4.0    1
dtype: int64
```

Частотный анализ:

```
[5] ser.mean() # среднее ариф.
```

```
2.6
```

```
[6] ser.std() # Ср.кв.отклонение
```

```
1.140175425099138
```

```
[7] ser.median() # медиана
```

```
3.0
```

```
[8] ser.quantile(3/4) # квантиль
```

```
3.0
```

```
ser.count() # кол-во эл-ов
```

```
5
```

Векторные операции

```
▶ a=pd.Series(data=[10,20,30,40] , index=["A","B","C","D"])
b=pd.Series(data=[1,2,3,4] , index=["A","B","C","D"])
a+b
```

```
↗
0
A  11
B  22
C  33
D  44
```

Применить какую-то функцию к каждому элементу серии – метод **apply**:

```
▶ b.apply(lambda x: x*100)
```

```
↗
0
A  100
B  200
C  300
D  400
```

НО! Нет inplace!

```
[12] a*b
```

```
↗
0
A  10
B  40
C  90
D  160
```

```
[13] -a+25
```

```
↗
0
A  15
B   5
C  -5
D -15
```

```
▶ a>20
```

```
↗
0
A  False
B  False
C   True
D   True
```

Массовое копирование

Значения из одной серии можно массово скопировать в другую серию методом **update**. Копирование будет происходить, основываясь на индексы-метки, а не на последовательность элементов. Для меток старой серии, в которой не найдены метки новой серии, будут оставлены старые значения.

```
a=pd.Series(data=[10,20,30,40] , index=["A","B","C","D"])
b=pd.Series(data=[1,2] , index=["Z","B"])
print(a)
a.update(b)
print(a)
```

```
A    10
B    20
C    30
D    40
dtype: int64
A    10
B     2
C    30
D    40
dtype: int64
```

Построение графиков на основе Series

На основе рядов Pandas можно выполнять построение различных графиков и диаграмм с помощью библиотеки matplotlib [5].

Для установки matplotlib можно использовать команду:

```
python.exe -m pip install matplotlib
```

График цен

Построим график цен в зависимости от их номера в ряду.

► In [6]:

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 fruits = ["Яблоки сезонные", "Груши «Конференция»",
5           "Бананы", "Мандарины (Турция)", "Апельсины"]
6 prices = [99, 145, 75, 90, 125]
7 fruits_series = pd.Series(prices)
8 fruits_series.plot()
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcb574ef4a8>

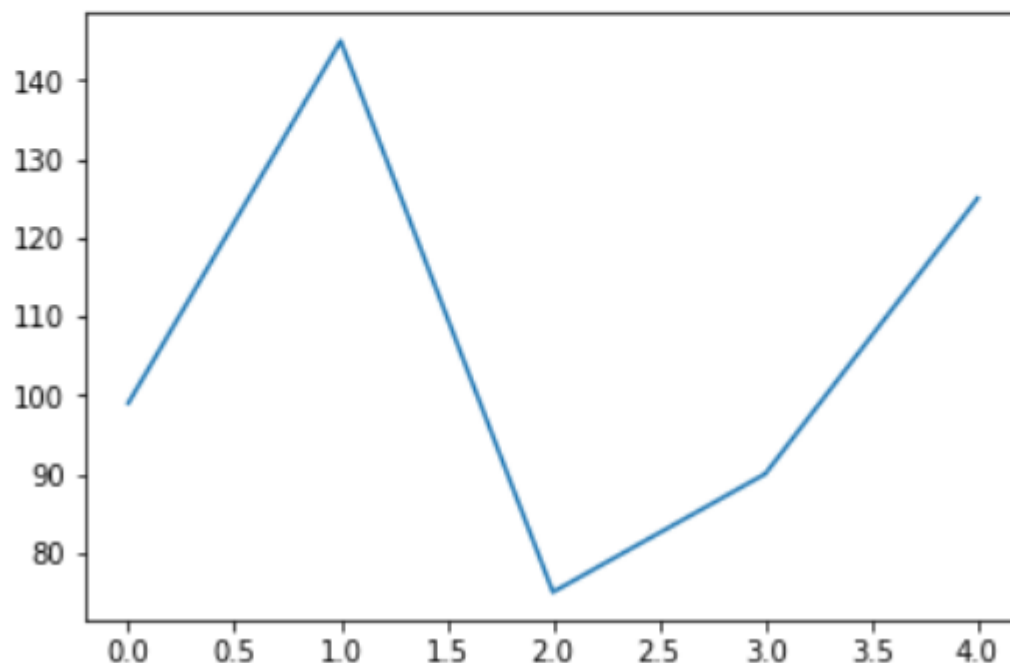


График цен

Построим график цен в зависимости от фруктов.

```
In [9]: 1 import pandas as pd
        2 import matplotlib.pyplot as plt
        3
        4 fruits = ["Яблоки сезонные", "Груши «Конференция»",
        5           "Бананы", "Мандарины (Турция)", "Апельсины"]
        6 prices = [99, 145, 75, 90, 125]
        7 fruits_series = pd.Series(prices, index = fruits)
        8 fruits_series.plot(xticks = range(len(fruits_series.index)),
        9                    use_index = True, rot = 90)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd75fcfc630>

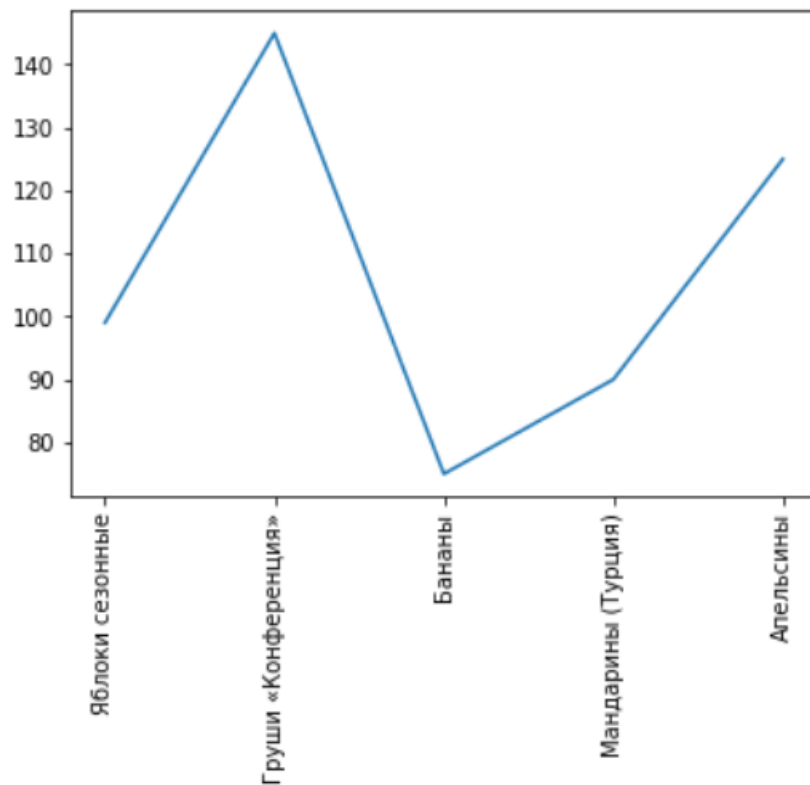


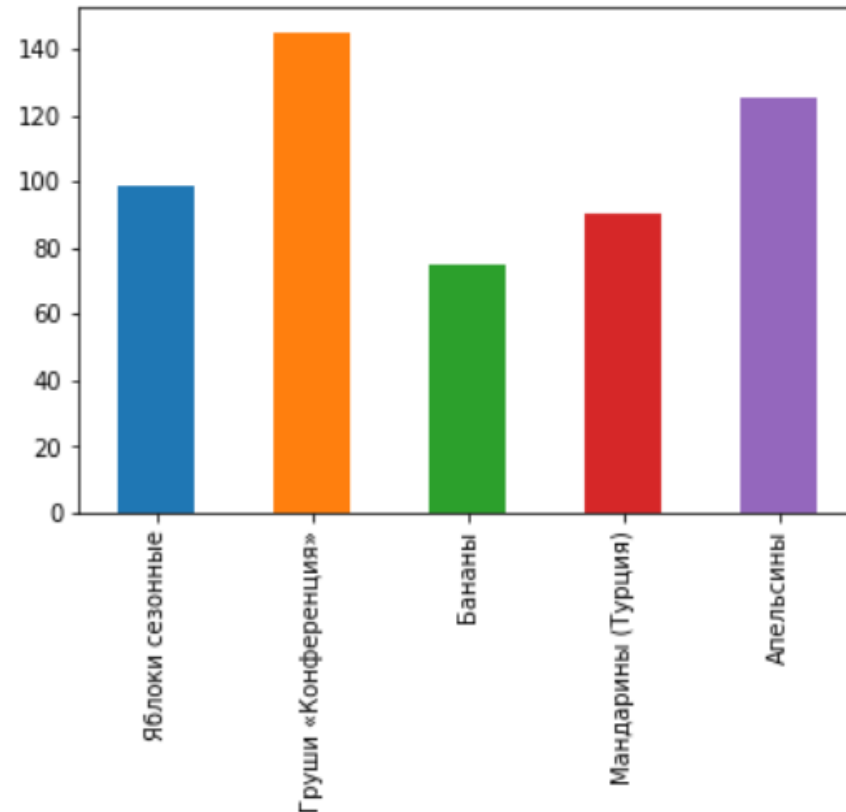
Диаграмма цен

Построим столбчатую диаграмму цен.

In [10]:

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 fruits = ["Яблоки сезонные", "Груши «Конференция»",
5           "Бананы", "Мандарины (Турция)", "Апельсины"]
6 prices = [99, 145, 75, 90, 125]
7 fruits_series = pd.Series(prices, index = fruits)
8 fruits_series.plot(kind = "bar")
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd75fdc4e80>



Часть 2

PANDAS. DATAFRAME

DataFrame

Pandas DataFrame – это тип данных предназначенный для представления информации, используемой в задача машинного обучения.

По своей сути – это двумерная изменяемая таблица, у которой координаты могут быть заменены на ключи произвольного типа (например, строки). Причём это могут быть как названия столбцов, так и названия строк!

Каждая колонка датафрейма – это ряд. Фактически, датафрейм – это словарь рядов.

Объекты класса DataFrame

```
df=pd.DataFrame({"A":[1,2,3], "B":[4,5,6], "C":[7,8,9]}, index=["f","s","t"])
df
```

	A	B	C
f	1	4	7
s	2	5	8
t	3	6	9



df.index

Index(['f', 's', 't'], dtype='object')

df.columns

Index(['A', 'B', 'C'], dtype='object')

Доступ к серия-столбцам:

df['A']

df.A

	A
f	1
s	2
t	3

Но! Через точку
нельзя создать
новый столбец!

```
df['D']=df.A*10+df.B*5+df.C
df
```

	A	B	C	D
f	1	4	7	37
s	2	5	8	53
t	3	6	9	69



Лос и iloc

Серия значений одной строки:

	A	B	C	D
f	1	4	7	37
s	2	5	8	53
t	3	6	9	69

```
df.loc['t']
```

t	
A	3
B	6
C	9
D	69

```
df.iloc[1:3]
```

	A	B	C	D
s	2	5	8	53
t	3	6	9	69

```
df.loc['f':'s']
```

	A	B	C	D
f	1	4	7	37
s	2	5	8	53

```
df.loc['f','A']
```

1

```
df.loc['f':'s','B':'C']
```

	B	C
f	4	7
s	5	8

```
df.iloc[:,::-1]
```

	D	C	B	A
f	37	7	4	1
s	53	8	5	2
t	69	9	6	3

Редукция столбцов и строк

```
df.sum() # по умолчанию axis=0 - свертка по вертикали  
df.sum(axis=1) # свертка по горизонтали  
df.max()  
df.idxmax()  
  
df.agg(['min', 'sum', 'mean'])
```

	A	B	C	D
min	1.0	4.0	7.0	37.0
sum	6.0	15.0	24.0	159.0
mean	2.0	5.0	8.0	53.0



Drop и dropna

Для удаления столбцов:

```
df2=df.drop(columns=['B','D'])  
df2
```

	A	C
f	1	7
s	2	8
t	3	9

Для удаления строк:

```
df3=df.drop(labels=['f'])  
df3
```

	A	B	C	D
s	2	5	8	53
t	3	6	9	69

```
df.dropna() # по умолчанию удаляет все строки с Nan
```

```
df.dropna(axis='columns') # удаляет все столбцы с Nan
```

Метод groupby

Данный метод осуществляет что-то вроде корзиной сортировки для последующей агрегации. Группировка происходит по уникальным значениям в указанном столбце. Эти значения будут играть роль индекса в новом листе.

```
f=pd.DataFrame({"name":["Аня","Петя","Катя"], "картошка":[4,5,6], "морковка":[7,8,9]}, index=["1","2","3"])  
f
```

	name	картошка	морковка
1	Аня	4	7
2	Петя	5	8
3	Катя	6	9

```
f.groupby("name").sum()
```

	картошка	морковка
name		
Аня	4	7
Катя	6	9
Петя	5	8

Слияние данных из нескольких таблиц

Конкатенация таблиц **concat** – механическое добавление строк второй таблицы к строкам первой.

```
frm3 = pd.DataFrame([['Иванов', 500000], ['Петров', 25000]],  
                    columns = ['fio', 'salary'])  
frm3  
frm4 = pd.DataFrame([['Иванов', 'audi'], ['Сидоров', 'lada']],  
                    columns = ['fio', 'car'])
```

	fio	salary
0	Иванов	500000
1	Петров	25000

	fio	car
0	Иванов	audi
1	Сидоров	lada

```
frm=pd.concat([frm3,frm4])  
frm
```

По вертикали

	fio	salary	car
0	Иванов	500000.0	NaN
1	Петров	25000.0	NaN
0	Иванов	NaN	audi
1	Сидоров	NaN	lada

По горизонтали

```
frm=pd.concat([frm3,frm4], axis=1)  
frm
```

	fio	salary	fio	car
0	Иванов	500000	Иванов	audi
1	Петров	25000	Сидоров	lada

Объединение с помощью merge

	fio	salary
0	Иванов	500000
1	Петров	25000

0	Иванов	500000
1	Петров	25000

	fio	car
0	Иванов	audi
1	Сидоров	lada

0	Иванов	audi
1	Сидоров	lada

```
frm3.merge(frm4)
```

	fio	salary	car
0	Иванов	500000	audi

0	Иванов	500000	audi
---	--------	--------	------

По умолчанию параметр how=inner

```
frm3.merge(frm4, how='left')
```

	fio	salary	car
0	Иванов	500000	audi
1	Петров	25000	NaN

0	Иванов	500000	audi
1	Петров	25000	NaN

```
frm3.merge(frm4, how='right')
```

	fio	salary	car
0	Иванов	500000.0	audi
1	Сидоров	NaN	lada

0	Иванов	500000.0	audi
1	Сидоров	NaN	lada

```
frm3.merge(frm4, how='outer')
```

	fio	salary	car
0	Иванов	500000.0	audi
1	Петров	25000.0	NaN
2	Сидоров	NaN	lada

0	Иванов	500000.0	audi
1	Петров	25000.0	NaN
2	Сидоров	NaN	lada

Полезные ссылки и литература

1. <https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range> – документация Python о последовательных типах данных
2. <https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset> – документация Python о множествах
3. <https://docs.python.org/3/library/stdtypes.html#mapping-types-dict> – документация Python о словарях
4. <https://pandas.pydata.org/> – сайт библиотеки Pandas
5. <https://matplotlib.org/> – сайт библиотеки Matplotlib