Machine Learning

Topic 2. Lecture 2

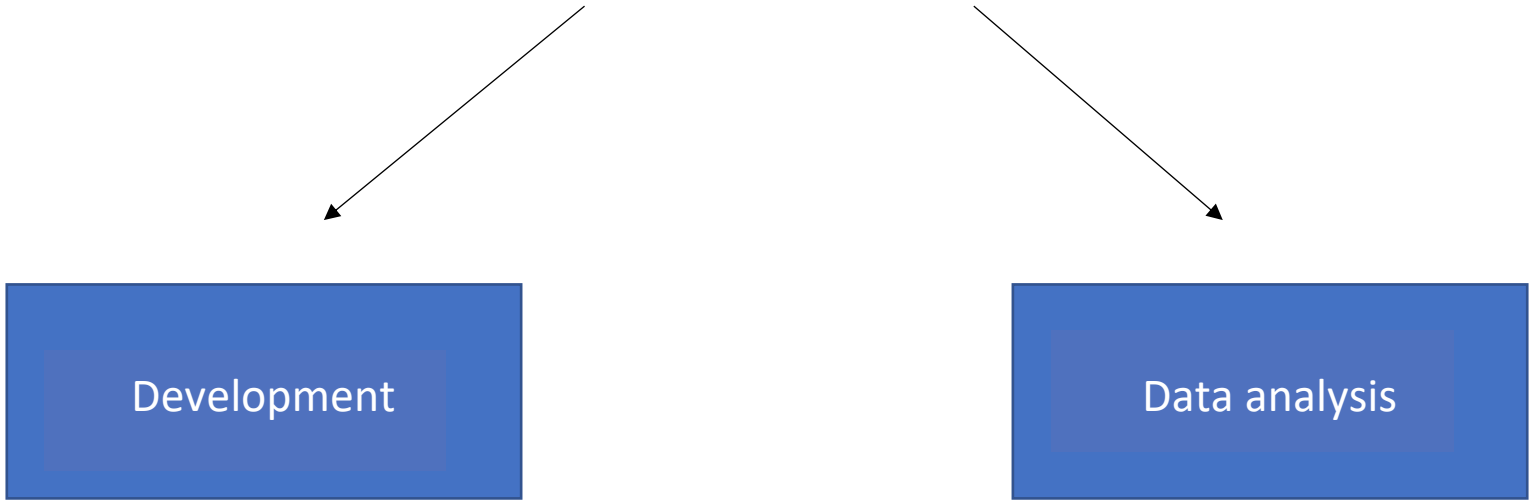
Python for Data Analysis. Libraries

Yury Sanochkin

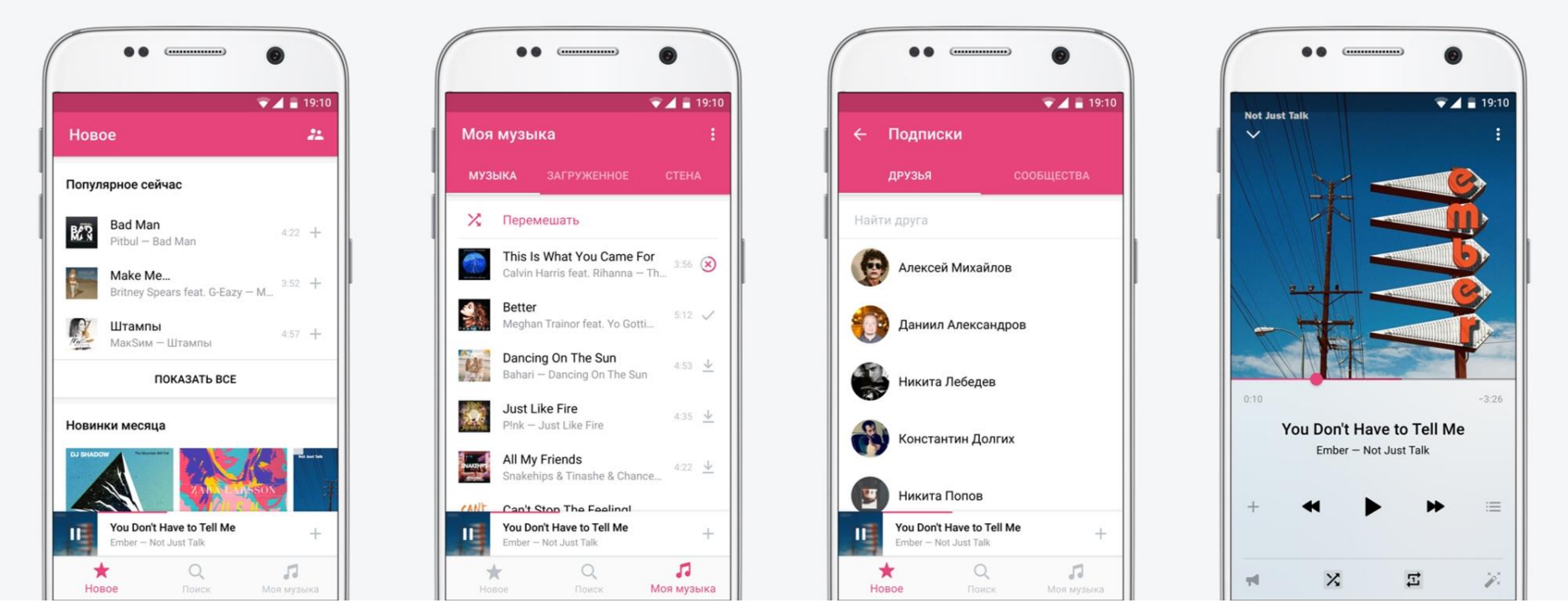
ysanochkin@hse.ru

NRU HSE, 2025

What are the types of programming?

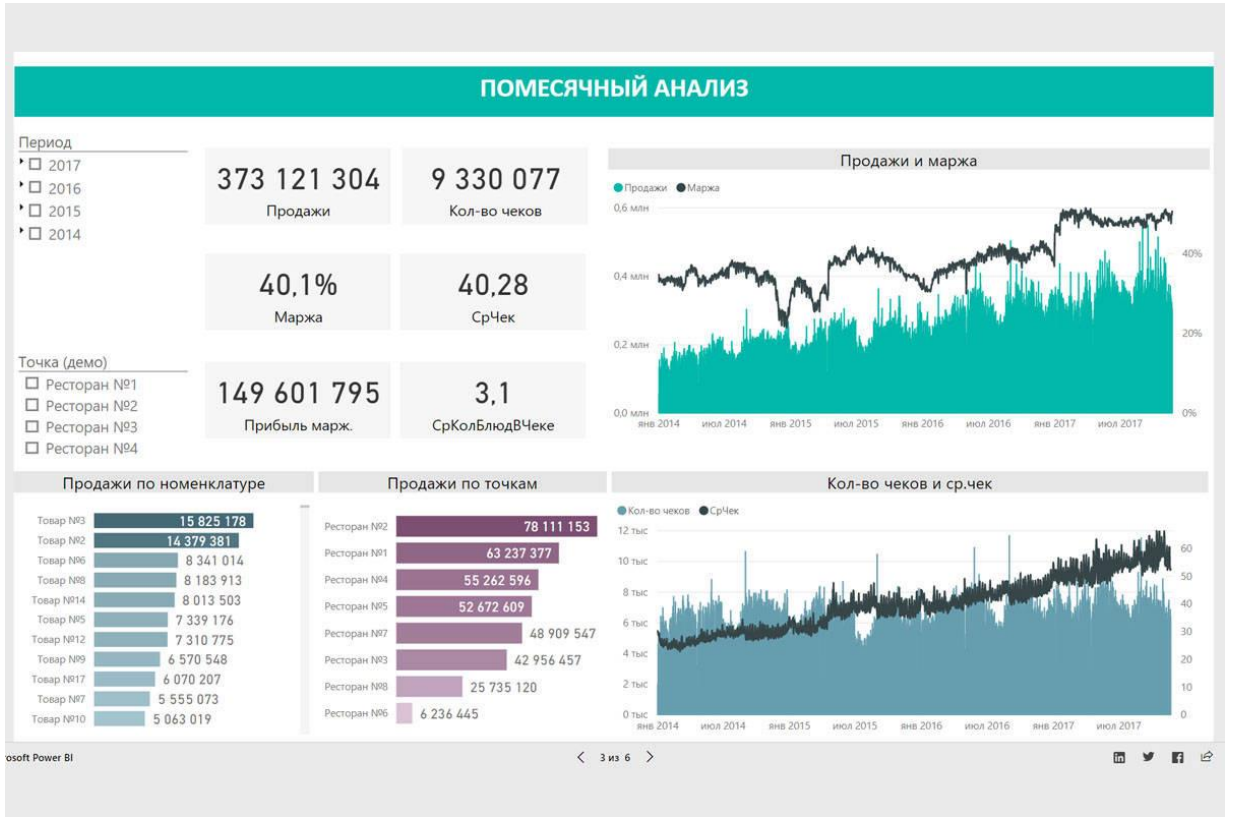


Could be like that



What is the goal? Result?

Could be even that



What is the goal and result here?

What special do we need for data analysis?

- What do you think a language should “be able to do” for data analysis tasks?

What special do we need for data analysis?

- What do you think a language should “be able to do” for data analysis tasks?
- There are many important requirements
- The most important thing is to be able to work quickly and efficiently with large amounts of data.
- Visualize the found dependencies

What special do we need for data analysis?

- What do you think a language should “be able to do” for data analysis tasks?
- There are many important requirements
- The most important thing is to be able to work quickly and efficiently with large amounts of data.
- Visualize the found dependencies
- Python itself doesn't quite know how to do this.
- But special libraries can!

Libraries in Python

- There are a huge number of libraries, including those designed for various types of data analysis
- The most key and common knowledge of which is necessary for any analyst:
  - Numpy – a library for efficiently working with matrices, vectors and other mathematical objects (very fast)
  - Pandas – a library for working with data presented as a matrix
  - Matplotlib – library for data visualization

Libraries’ import in Python

- What are the ways to import a library?

Libraries’ import in Python

- What are the ways to implement functionality from the library?

1) Just import the library

```
[1] import math

    sinus = math.sin(math.pi / 2)
    print(sinus)
```

```
1.0
```

Libraries' import in Python

- What are the ways to implement functionality from the library?

2) Import the necessary functions from the library

```
[2]  from math import sin, pi

     sinus = sin(pi / 2)
     print(sinus)
```

```
     1.0
```

Libraries' import in Python

- What are the ways to implement functionality from the library?

3) Import everything from the library (bad option)

```
[3]  from math import *

     sinus = sin(pi / 2)
     print(sinus)

     1.0
```

Why is it bad?

Libraries' import in Python

- Basic libraries for data analysis are usually imported in the first way, using "common" abbreviations

```
[2]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
```

Numpy Library

Numpy Library

- Designed for efficient work with multidimensional vectors, matrices and other mathematical objects
- Very fast. Because it is not entirely written in pure Python

Numpy Library

- Designed for efficient work with multidimensional vectors, matrices and other mathematical objects
- Very fast. Because it is not entirely written in pure Python
- The main object is a Numpy Array
- Numpy Array has the key feature of Array from C-like languages: it can store objects of only one type (very fast due to this)
- It can also be dynamically expanded

Numpy Array

- Simple one-dimensional vector

```
[2]  vec = np.array([1, 2, 3])

     vec

     array([1, 2, 3])

[3]  type(vec)

     numpy.ndarray
```

Numpy Array

- What do you think, how to set the matrix?

Numpy Array

- What do you think, how to set the matrix?
- Array of arrays

```
[4]  vec2 = np.array([[1, 2, 3], [4, 5, 6]])

     vec2

     array([[1, 2, 3],
            [4, 5, 6]])

[5]  type(vec2)

     numpy.ndarray
```

Numpy Array

- You can also have a three-dimensional object...

```
[9]  vec3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

     vec3

     array([[[1, 2],
             [3, 4]],

            [[5, 6],
             [7, 8]]])

[10] type(vec3)

     numpy.ndarray
```

Dimensions in Numpy

- You can view the dimension using the **shape** function
- What dimension will vec, vec2, vec3 from the examples above have?

```
[ ]  print(vec.shape, vec2.shape, vec3.shape)
```

Dimensions in Numpy

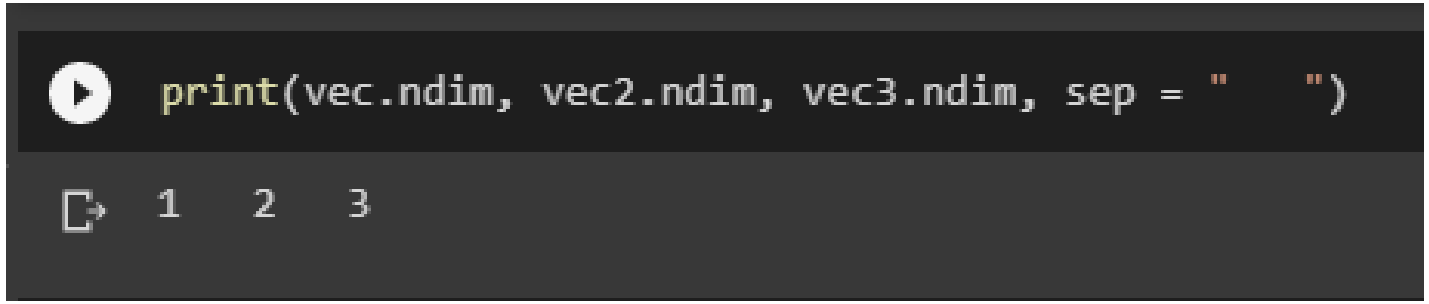
- You can view the dimension using the **shape** function
- What dimension will vec, vec2, vec3 from the examples above have?

```
[22] print(vec.shape, vec2.shape, vec3.shape)

     (3,) (2, 3) (2, 2, 2)
```

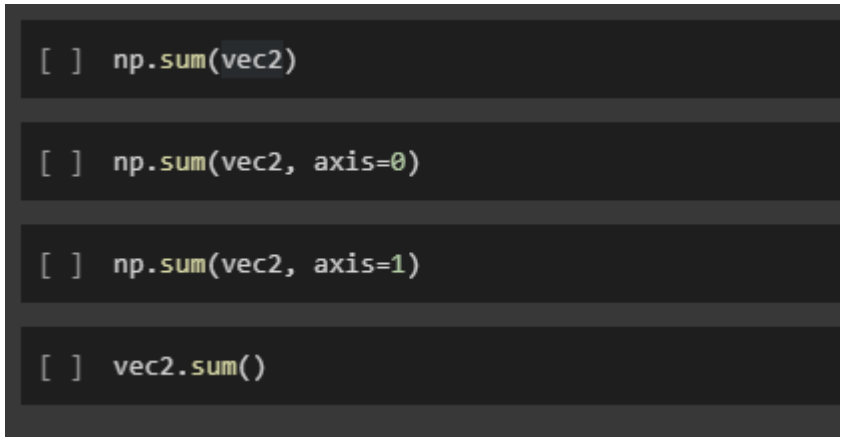
Dimensions in Numpy

- You can also view the number of axes using the **ndim** function



Dimensions in Numpy

- Some functions have an axis parameter, which allows you to apply this function along different axes - in this case, along rows or columns
- What will this code output?



Dimensions in Numpy

```
[25] np.sum(vec2)

     21

[26] np.sum(vec2, axis=0)

     array([5, 7, 9])

[27] np.sum(vec2, axis=1)

     array([ 6, 15])

[28] vec2.sum()

     21
```

Dimensions in Numpy

- Finally, another important function related to dimensions is **reshape()**
- Allows you to change the dimension of the vector, which can sometimes be very useful
- What will this code output?

```
[ ]  vec2.reshape(3, 2)

[ ]  vec2.reshape(-1, 2)

[ ]  vec2.reshape(3, -1)
```

Dimensions in Numpy

```
[29] vec2

     array([[1, 2, 3],
            [4, 5, 6]])

[30] vec2.reshape(3, 2)

     array([[1, 2],
            [3, 4],
            [5, 6]])

[31] vec2.reshape(-1, 2)

     array([[1, 2],
            [3, 4],
            [5, 6]])

[32] vec2.reshape(3, -1)

     array([[1, 2],
            [3, 4],
            [5, 6]])
```

Slicing in Numpy

- Works very similarly to regular slices in Python, but operates on each axis (the axes are separated by commas)

```
[23] vec2 = vec2.reshape(3, 2)
     vec2

     array([[1, 2],
            [3, 4],
            [5, 6]])

[19] vec2[:, 1]

[20] vec2[2, :]

[21] vec2[1:2, 0]

[22] vec2[::2, :]
```

Slicing in Numpy

```
[23] vec2 = vec2.reshape(3, 2)
     vec2

     array([[1, 2],
            [3, 4],
            [5, 6]])

[24] vec2[:, 1]

     array([2, 4, 6])

[25] vec2[2, :]

     array([5, 6])

[26] vec2[1:2, 0]

     array([3])

[27] vec2[::2, :]

     array([[1, 2],
            [5, 6]])
```

Operations with matrices

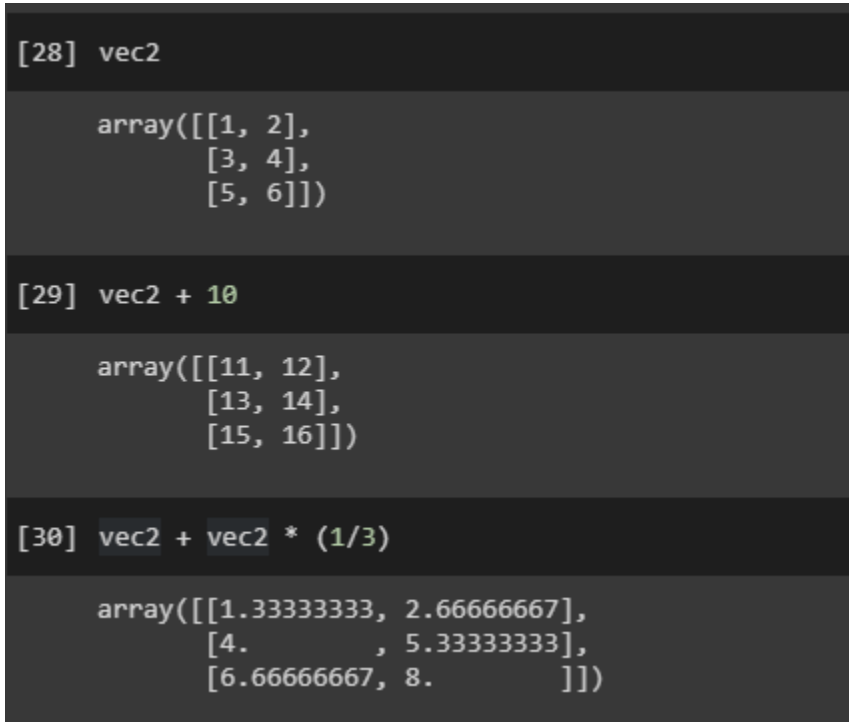
- Let's remember lineal algebra! ☺
- What operations on matrices do you remember?

Operations with matrices

- Let's remember lineal algebra! ☺
- What operations on matrices do you remember?
- Matrix addition
- Multiplying a matrix by a number
- Matrix-matrix multiplication
- Matrix transpose
- There may be more complex operations!

Operations with matrices

- Numpy implements extremely convenient work with matrices, in terms of operations. Take a look:



Operations with matrices

- Even like that:

```
[31] np.sin(vec2) + vec2 ** 2

     array([[ 1.84147098,  4.90929743],
            [ 9.14112001, 15.2431975 ],
            [24.04107573, 35.7205845 ]])
```

Operations with matrices

- Even like that:

```
[31] np.sin(vec2) + vec2 ** 2

     array([[ 1.84147098,  4.90929743],
            [ 9.14112001, 15.2431975 ],
            [24.04107573, 35.7205845 ]])
```

- In fact, you can immediately apply any mathematical function to the matrix

Boolean arrays

- And one more extremely important functionality that appeared in Numpy - Boolean arrays
- In fact, they allow you to implement full-fledged data filters, and will be very useful in the future

Boolean arrays

- And one more extremely important functionality that appeared in Numpy - Boolean arrays
- In fact, they allow you to implement full-fledged data filters, and will be very useful in the future
- What will this code output?

```
[ ]  is_even = vec2 % 2 == 0
     is_even
```

Boolean arrays

- And one more extremely important functionality that appeared in Numpy - Boolean arrays
- In fact, they allow you to implement full-fledged data filters, and will be very useful in the future
- What will this code output?

```
[33] is_even = vec2 % 2 == 0
     is_even

     array([[False,  True],
            [False,  True],
            [False,  True]])
```

Boolean arrays

- One more example

```
[34] new_vec = np.array([[1, 7], [8, 9], [-2, 0], [11, 3]])

[36] is_even = new_vec % 2 == 0
     is_even

     array([[False, False],
            [ True, False],
            [ True,  True],
            [False, False]])
```

Boolean arrays

- And now the most important feature
- What do you think this code does?

```
[ ]  new_vec[new_vec % 2 == 0]
```

Boolean arrays

- And now the most important feature
- What do you think this code does?
- Data filtering!

```
[34] new_vec = np.array([[1, 7], [8, 9], [-2, 0], [11, 3]])

[37] new_vec[new_vec % 2 == 0]

     array([ 8, -2,  0])
```

Boolean arrays

- And now the most important feature
- What do you think this code does?
- Data filtering!

```
[34] new_vec = np.array([[1, 7], [8, 9], [-2, 0], [11, 3]])

[37] new_vec[new_vec % 2 == 0]

     array([ 8, -2,  0])
```

Pandas Library and Matplotlib Library

Pandas Library and Matplotlib Library

- Numpy is about numbers and mathematical objects, and Pandas is about data
- Although the approach to working with tables is almost exactly the same as in Numpy
- Matplotlib is about graphics and visualization

Pandas Library and Matplotlib Library

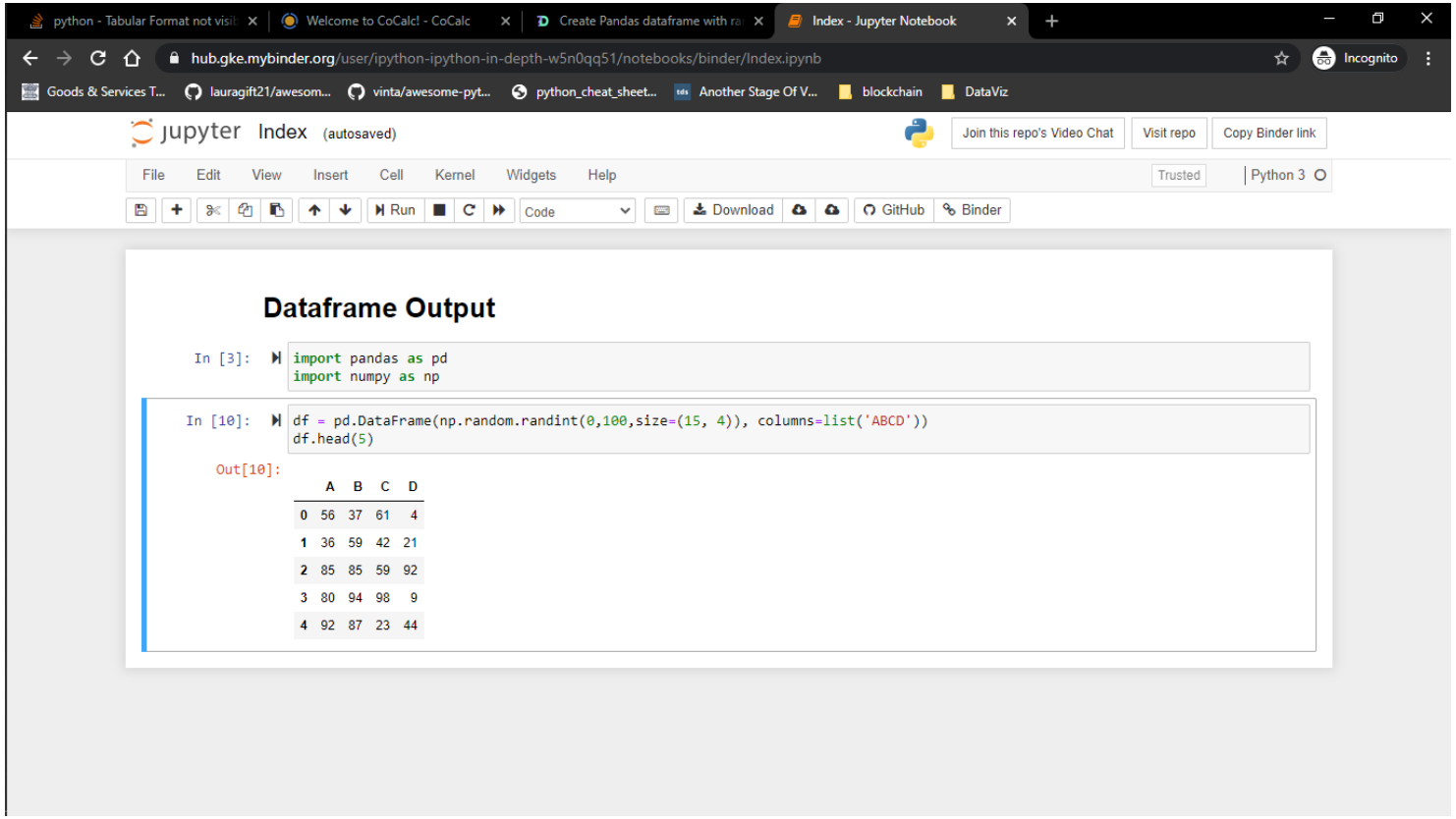
- Where will we work with them?

Pandas Library and Matplotlib Library

- Where will we work with them?
- While it was still possible for Numpy to work in classic Python programming environments, it is almost impossible to beautifully visualize data and especially graphs in standard console Input/Output

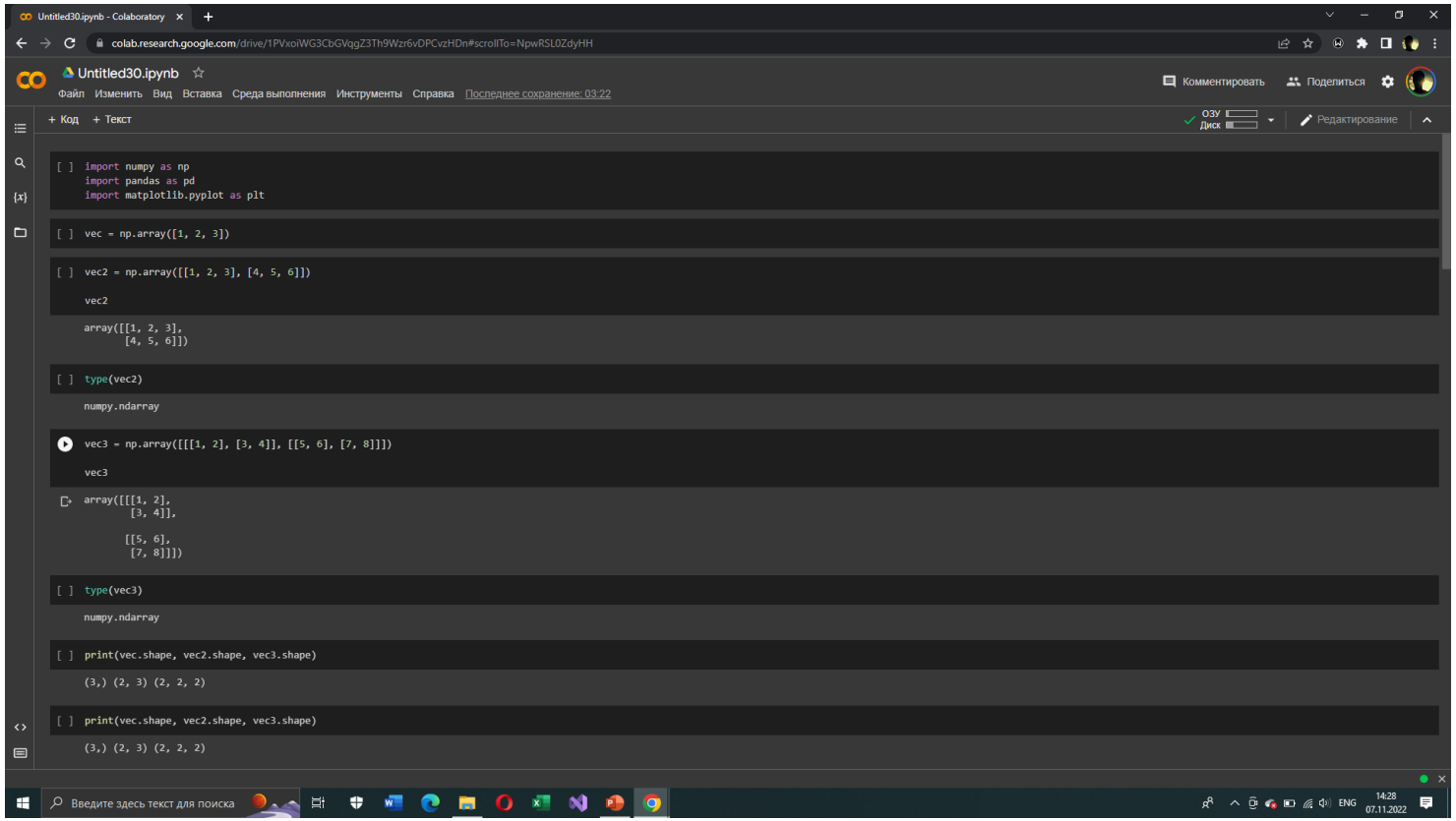
Jupyter Notebook и Google Colab

- There are 2 working options: Jupyter Notebook (Anaconda)



Jupyter Notebook и Google Colab

- And a free online environment from Google - Google Collaboratory



Jupyter Notebook и Google Colab

- We will learn more about these environments at seminars.
- They allow you to remember (for some time) a large number of local variables, restart only some parts of the code, and much more
- Including, they allow you to draw graphs, write text comments, formulas, and make visualizations

Data in Pandas

- Let's download data from the Titanic survivors dataset (perhaps the most famous dataset in the field of data analysis training)
- You will still have to work with this dataset in HW №3 ☺

Data in Pandas

```
[8]  df = pd.read_csv("train (1).csv", sep=',')
```

```
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Basic types in Pandas

- There are two main data types in Pandas: Pandas DataFrame and Pandas Series
- The tables themselves (including the one presented on the last slide) are of the Pandas DataFrame type

```
[10] type(df)

     pandas.core.frame.DataFrame
```

Basic types in Pandas

- Pandas Series is a one-dimensional vector table slice
- Column slice

```
[20] column = df['Name']
     column
```

```
0                                Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                Heikkinen, Miss. Laina
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                              Allen, Mr. William Henry
                              ...
886                              Montvila, Rev. Juozas
887                       Graham, Miss. Margaret Edith
888           Johnston, Miss. Catherine Helen "Carrie"
889                              Behr, Mr. Karl Howell
890                                Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

```
[19] type(column)
```

```
pandas.core.series.Series
```

Basic types in Pandas

- String slicing (less common thing in Pandas)

```
[16] line = df.iloc[5]
     line
```

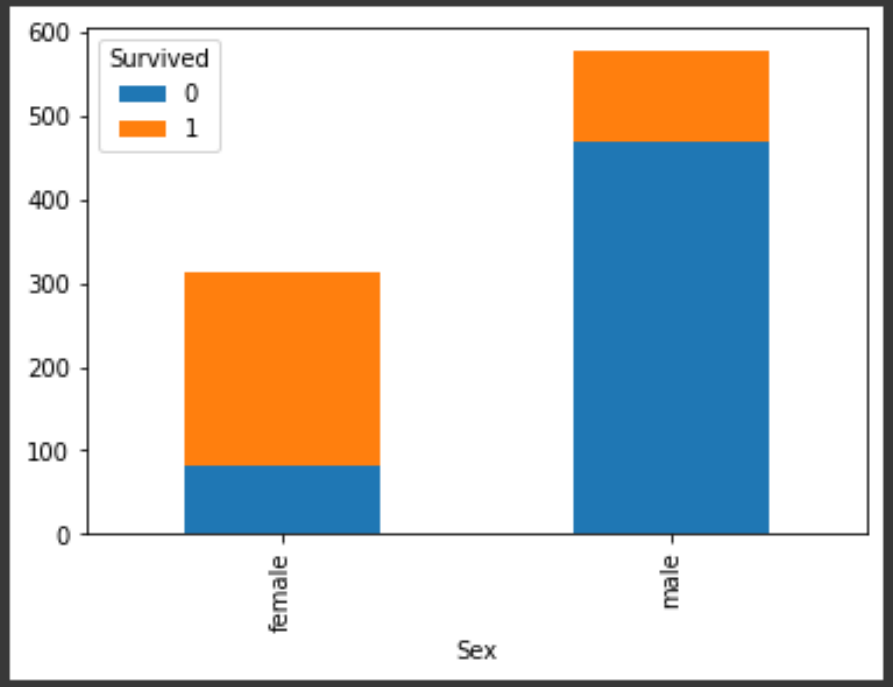
```
PassengerId                  6
Survived                     0
Pclass                       3
Name           Moran, Mr. James
Sex                       male
Age                        NaN
SibSp                        0
Parch                        0
Ticket                  330877
Fare                    8.4583
Cabin                      NaN
Embarked                     Q
Name: 5, dtype: object
```

```
type(line)
```

```
pandas.core.series.Series
```

Let’s look at some visualization at the end!

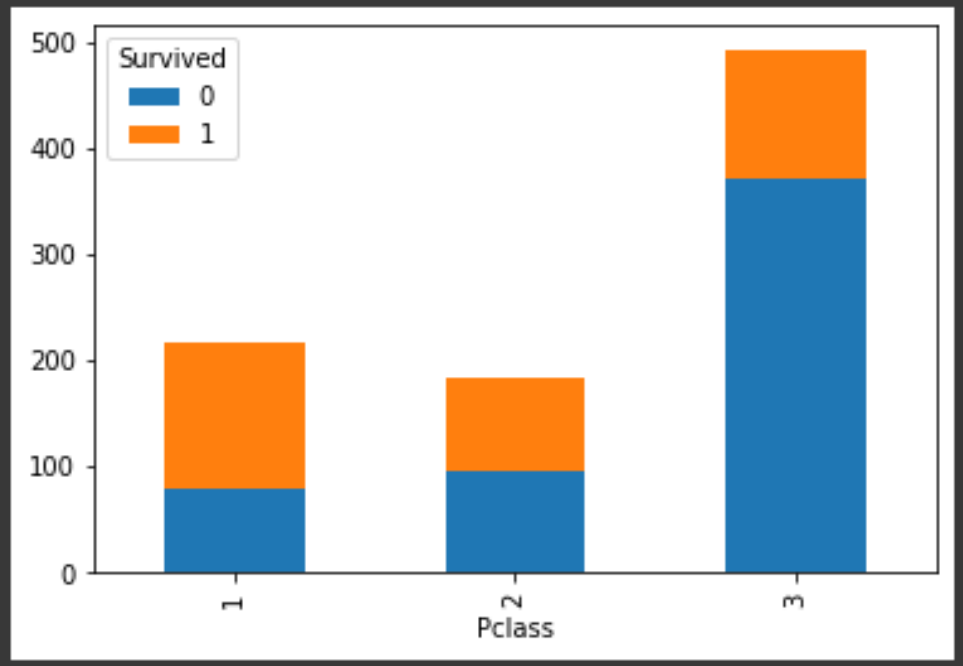
```
[22] df.pivot_table('PassengerId', 'Sex', 'Survived', 'count').plot(kind='bar', stacked=True);
```



What conclusions can be drawn?

Let’s look at some visualization at the end!

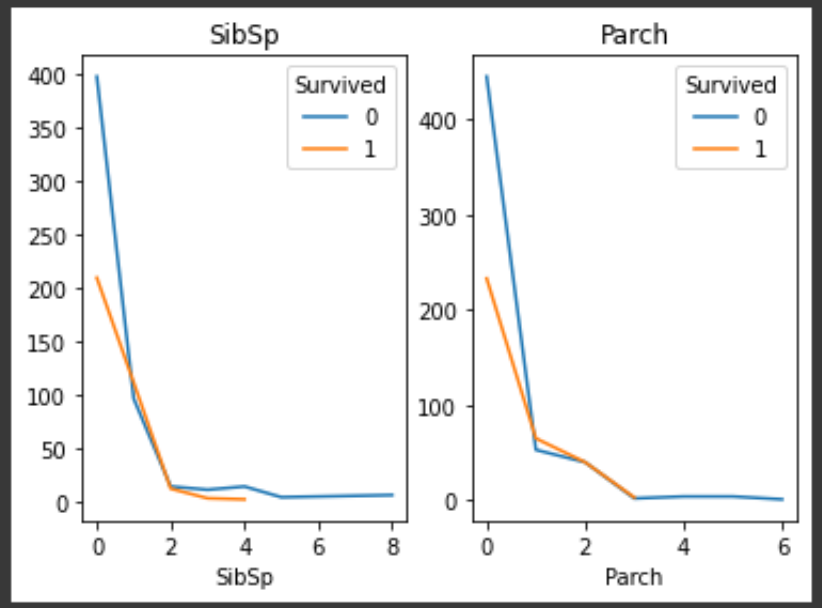
```
[23] df.pivot_table('PassengerId', 'Pclass', 'Survived', 'count').plot(kind='bar', stacked=True);
```



What conclusions can be drawn?

Let’s look at some visualization at the end!

```
[24] fig, axes = plt.subplots(ncols=2)
     df.pivot_table('PassengerId', ['SibSp'], 'Survived', 'count').plot(ax=axes[0], title='SibSp');
     df.pivot_table('PassengerId', ['Parch'], 'Survived', 'count').plot(ax=axes[1], title='Parch');
```



What conclusions can be drawn?