

# Лабораторная работа № 1

## Цели и задачи

Цель изучение инструментов интеллектуальной обработки данных (Google Colab)

### Задачи

1. Изучение Google Colab
2. Загрузка данных из датасета

Датасеты

<https://www.kaggle.com/datasets>

<https://datasetsearch.research.google.com/>

<https://github.com/awesomedata/awesome-public-datasets>

3. Изучение
  - а. построение статистики
  - б. построение графиков
4. Создание аккаунта и размещение проекта на GitLab

Блокнот Colab — это бесплатная интерактивная облачная среда для работы с кодом от Google. Принцип у нее такой же, как у остальных онлайн-офисов компании: она позволяет одновременно с коллегами работать с данными.

Collab построен на подобию Jupiter notebook, инструмента, который позволяет запускать Python код и сразу демонстрировать результат работы конкретного блока кода. У коллаба есть преимущество перед этой программой. Его код может выполняться не на самом компьютере, а на серверах Google. Есть бесплатный вариант пользоваться ими, но результаты работы сохраняются только на протяжении 12 часов. Кроме того, Google отключает блокноты после примерно 30 минут бездействия, чтобы не перегружать процессоры. Есть и платная подписка, в которой нет этих ограничений.

Главная особенность «Колаборатории» — бесплатные мощные графические процессоры GPU и TPU, благодаря которым можно заниматься не только

базовой аналитикой данных, но и более сложными исследованиями в области машинного обучения. С тем, что CPU вычисляет часами, GPU или TPU справляются за минуты или даже секунды.

- CPU — центральный процессор — мозг компьютера, который выполняет операции с данными. Настолько универсален, что может использоваться почти для всех задач: от записи фотографий на флешку до моделирования физических процессов.
- GPU — графический процессор. Обрабатывает данные быстрее, так как задачи выполняет параллельно, а не последовательно, как CPU. Он заточен исключительно под графику, поэтому на нем удобнее работать с изображением и видео, например заниматься 3D-моделированием или монтажом.
- TPU — тензорный процессор, разработка Google. Он предназначен для тренировки нейросетей. У этого процессора в разы выше производительность при больших объемах вычислительных задач.

---

С любым языком программирования в разы сложнее работать без библиотек. Чтобы узнать, с какими библиотеками поставляется коллаб, есть отдельная команда:

Быстро освоить или подтянуть свои знания в Python можно в следующих ресурсах:

- <https://www.learnpython.org/>
- <https://www.codecademy.com/>
- <https://checkio.org/>
- <https://stepik.org/course/67/>
- <https://stepik.org/course/512/>
- <https://stepik.org/course/431/>
- <https://stepik.org/course/56391/>

Мануалы по Google Collab:

- <https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c>

- <https://www.geeksforgeeks.org/how-to-use-google-colab/>
- Многое объясняющий обзорный блокнот:  
[https://colab.research.google.com/notebooks/basic\\_features\\_overview.ipynb](https://colab.research.google.com/notebooks/basic_features_overview.ipynb)

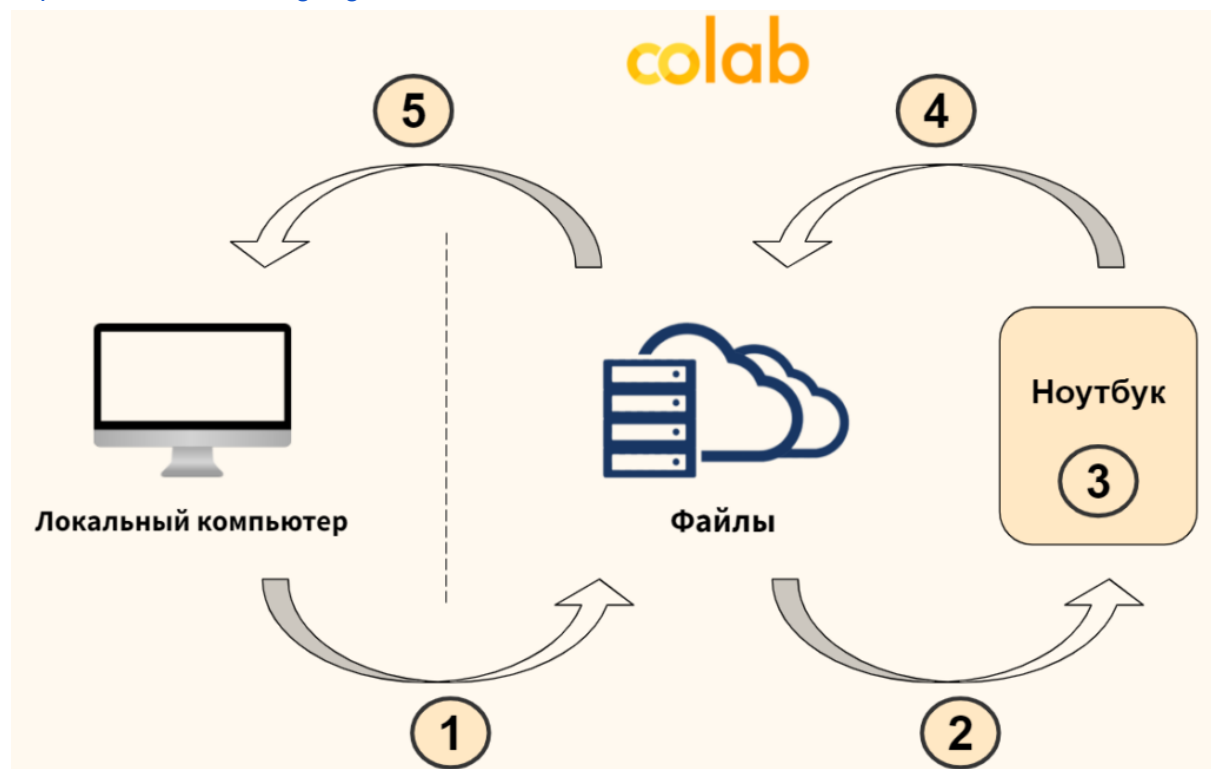
Мануалы по Jupiter:

- <https://jupyter-notebook.readthedocs.io/en/stable/>

Ход выполнения работы

Начало работы

<https://colab.research.google.com/>



В целом работа с файлами в Google Colab состоит из следующих этапов.

- **Этап 1.** Подгрузка файлов с локального компьютера на сервер Google

- **Этап 2.** Чтение файла
- **Этап 3.** Построение модели и прогноз
- **Этап 4.** Сохранение результата в новом файле на сервере Google
- **Этап 5.** Скачивание обратно на жесткий диск

Пройдемся по каждому из них. Но прежде поговорим про данные.

## Датасет «Титаник»

На этом занятии предлагаю взять датасет о пассажирах корабля «Титаник», который, как известно, затонул в 1912 году при столкновении с айсбергом. Часть пассажиров выжила, но многие, к сожалению, погибли. В предложенном датасете собрана информация о самих пассажирах (признаки), а также о том, выжили они или нет (целевая переменная).



Данные уже разделены на обучающую и тестовую выборки. Скачаем их.

[train.csv](#) скачать

[test.csv](#) скачать

Датасеты для самостоятельной работы

<https://www.kaggle.com/datasets>

<https://datasetsearch.research.google.com/>

<https://github.com/awesomedata/awesome-public-datasets>

# Этап 1. Подгрузка файлов в Google Colab

Если внешние файлы хранятся на локальном компьютере, то нам нужно подгрузить их в так называемое «Сессионное хранилище» (session storage, по сути, сервер Google).

Слово «сессионное» указывает на то, что данные, как я уже говорил, хранятся временно и после завершения очередной сессии стираются.

Подгрузить данные с локального компьютера можно двумя способами.

## Способ 1. Вручную через вкладку «Файлы»

Этот способ мы использовали до сих пор. В качестве напоминания приведу скриншоты подгрузки файла train.csv.

CO 04.Files.ipynb ☆

File Edit View Insert Runtime Tools Help Last edited on November 11

Files

(1) Откройте вкладку с файлами

(2) Выберите иконку загрузки файлов

▼ Этап 1. Подгрузка файлов

Способ 1. Вручную через вкладку "Файлы"

[ ] # см. материалы урока на сайте

Способ 2. Через модуль files библиотеки google.colab

```
# из библиотеки google.colab импортируем класс files
from google.colab import files

# создаем объект этого класса, применяем метод upload
uploaded = files.upload()
```

Открытие

← → ↕ ↑ > Этот компьютер > Рабочий стол

Поиск: Рабочий стол

Упорядочить ▾ Новая папка

Имя	Дата изменения
test.csv	11.11.2021 9:44
train.csv	11.11.2021 9:44

(3) Выберите файл **train.csv** и нажмите "Открыть"

Имя файла: train.csv All Files (\*.\*)

Открыть Отмена

## Способ 2. Через объект `files` библиотеки `google.colab`

К объекту `files` мы применяем метод `.upload()`, который передает нам словарь. Ключами этого словаря будут названия файлов, а значениями — сами загруженные данные. Приведем пример с файлом `test.csv`.

```
1 # из библиотеки google.colab импортируем класс files
2 from google.colab import files
3
4 # создаем объект этого класса, применяем метод
5 .upload()
   uploaded = files.upload()
```

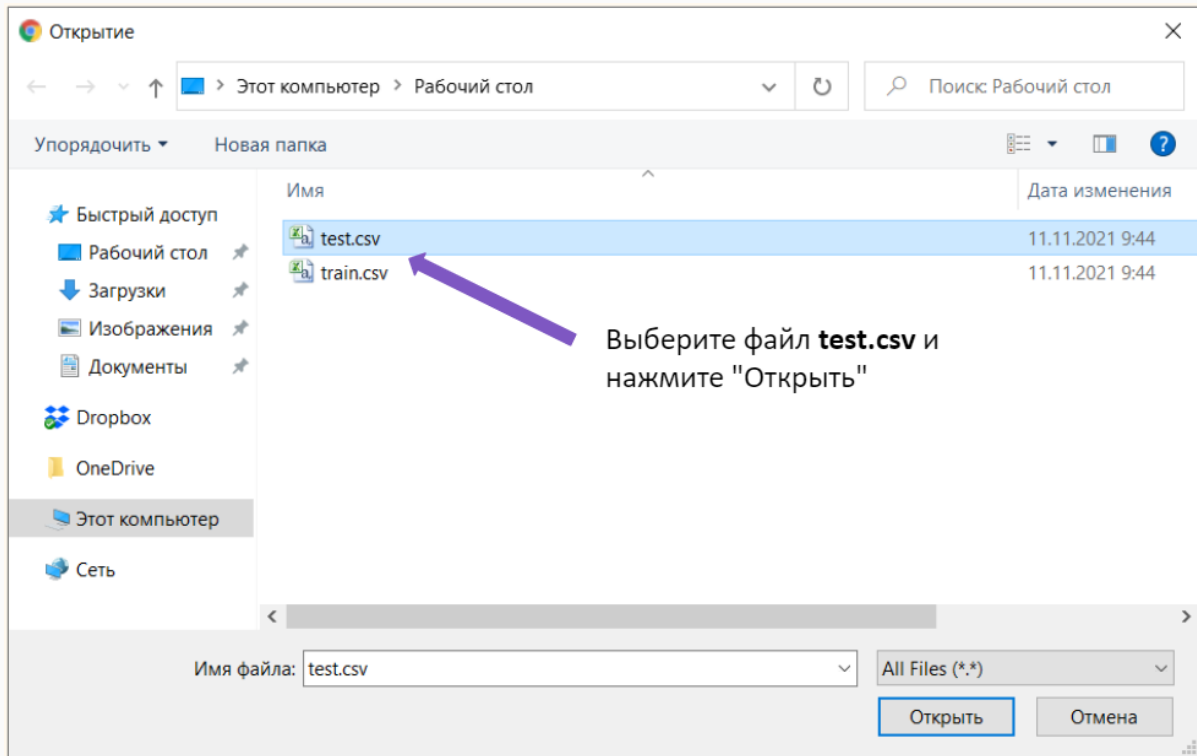
Нам будет предложено выбрать файл на жестком диске.



```
# создаем объект этого класса, применяем метод .upload()  
uploaded = files.upload()
```

Choose Files No file chosen

Cancel upload



```
1 # посмотрим на содержимое переменной uploaded
```

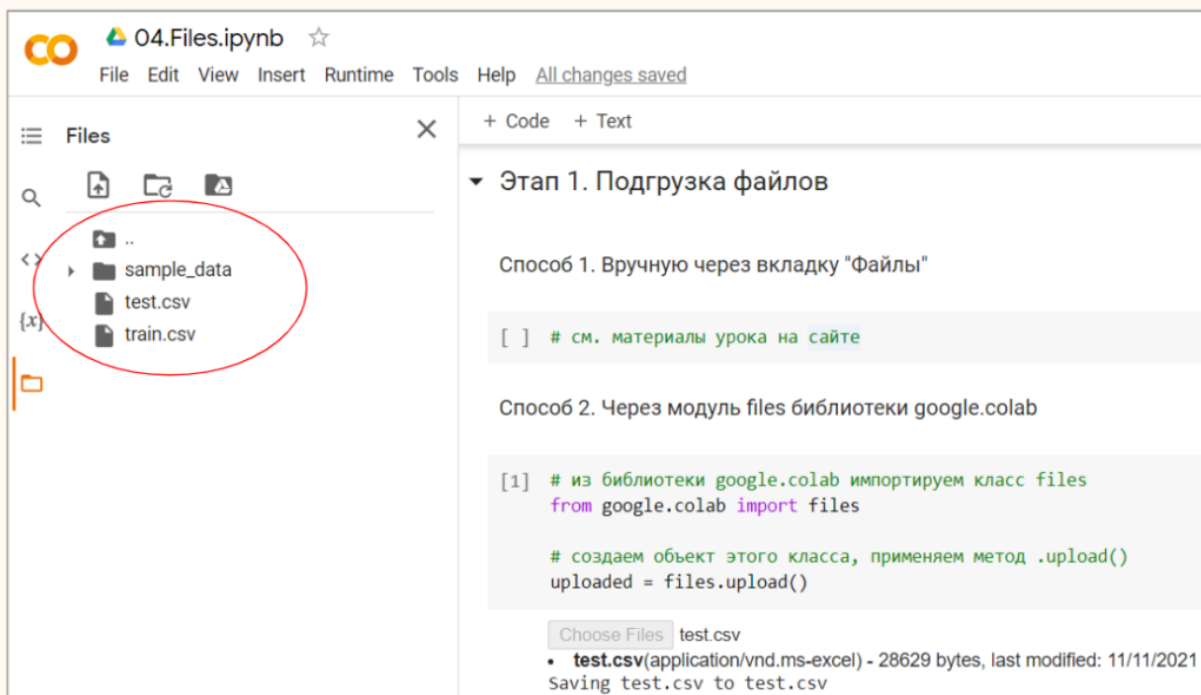
```
2 uploaded
```

```
1 {'test.csv':  
  b'PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,F  
  are,Cabin,Embarked\r\n892,3,"Kelly, Mr.  
  James",male,34.5,0,0,330911,7.8292,,Q\r\n893,3,"Wilkes  
  , Mrs. James (Ellen  
  Needs)",female,47,1,0,363272,7,,S\r\n894,2,... }'
```

Все что идет после двоеточия ( :) и есть наш файл. Он содержится в формате bytes, о чем свидетельствует буква b перед строкой файла (подробнее об этом ниже).

## Этап 2. Чтение файлов

После загрузки оба файла (train.csv и test.csv) оказываются в сессионном хранилище в папке под названием **/content/**.



The screenshot shows the Google Colab interface for a notebook titled "04.Files.ipynb". The left sidebar displays the file explorer with a folder named "sample\_data" containing two files, "test.csv" and "train.csv", which are circled in red. The main area shows the notebook content with two methods for uploading files. Method 1 is manual upload via the "Files" tab. Method 2 uses the "files" module from "google.colab". The code for Method 2 is as follows:

```
[1] # из библиотеки google.colab импортируем класс files
from google.colab import files

# создаем объект этого класса, применяем метод .upload()
uploaded = files.upload()
```

Below the code, a "Choose Files" button is shown next to "test.csv". The output of the code execution is displayed below:

- **test.csv**(application/vnd.ms-excel) - 28629 bytes, last modified: 11/11/2021

The status "Saving test.csv to test.csv" is shown at the bottom.

# Просмотр содержимого в папке /content/

## Модуль os и метод .walk()

Для того чтобы просмотреть ее содержимое *внутри ноутбука*, мы можем воспользоваться модулем **os** (отвечает за взаимодействие Питона с операционной системой) и, в частности, методом **.walk()** (позволяет «пройтись» по содержимому конкретной папки).

```
1  # импортируем модуль os
2
3
4  # выводим пути к папкам (dirpath) и наименования
   # файлов (filenames) и после этого
5  for dirpath, _, filenames in os.walk('/content/'):
6
7
8      # во вложенном цикле проходимся по названиям файлов
9
10     for filename in filenames:
11
12         # и соединяем путь до папок и входящие в эти папки
        # файлы
13
14         # с помощью метода path.join()
15
16     print(os.path.join(dirpath, filename))
17
18
```

```
1 /content/test.csv
2 /content/train.csv
3 /content/.config/gce
4 /content/.config/.last_update_check.json
5 /content/.config/config_sentinel
6 /content/.config/.last_opt_in_prompt.yaml
7 /content/.config/active_config
8 /content/.config/.last_survey_prompt.yaml
9 /content/.config/logs/2021.11.01/13.34.55.836922.log
10 /content/.config/logs/2021.11.01/13.34.28.082269.log
11 /content/.config/logs/2021.11.01/13.34.08.637862.log
12 /content/.config/logs/2021.11.01/13.34.55.017895.log
13 /content/.config/logs/2021.11.01/13.33.47.856572.log
14 /content/.config/logs/2021.11.01/13.34.35.080342.log
15 /content/.config/configurations/config_default
16 /content/sample_data/anscombe.json
17 /content/sample_data/README.md
18 /content/sample_data/california_housing_test.csv
19 /content/sample_data/mnist_train_small.csv
20 /content/sample_data/california_housing_train.csv
21 /content/sample_data/mnist_test.csv
```

Первые два файла и есть наши данные. В скрытой подпапке `/.config/` содержатся служебные файлы, а в подпапке `/sample_data/` — примеры датасетов, хранящиеся в Google Colab по умолчанию.

## Команда `!ls`

Кроме того, если нас интересуют только видимые файлы и папки, мы можем воспользоваться **командой `!ls`** (`ls` означает *to list*, т.е. «перечислить»).

```
1 !ls
```

```
1 sample_data test.csv train.csv
```

Подобным образом мы можем заглянуть внутрь папки `sample_data`.

```
1 !ls /content/sample_data/
```

```
1 anscombe.json          mnist_test.csv
```

```
2 california_housing_test.csv  mnist_train_small.csv
```

```
3 california_housing_train.csv README.md
```

Теперь прочитаем файл сначала из переменной `uploaded`, а затем напрямую из папки `/content/`.

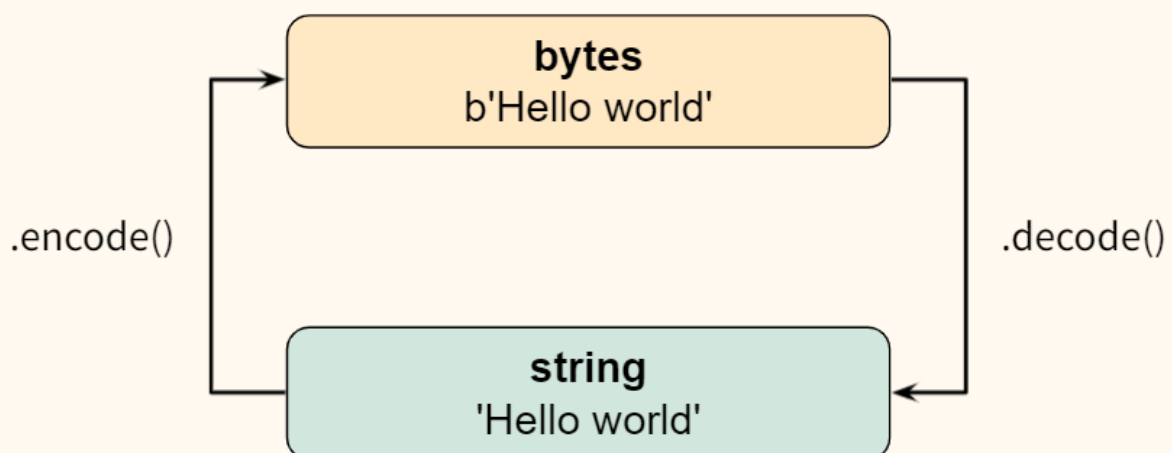
## Чтение из переменной `uploaded`

Как мы уже сказали выше, в словаре **`uploaded`** файл содержится в формате `bytes`.

```
1 # посмотрим на тип значений словаря uploaded
2
3 type(uploaded['test.csv'])
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Основная особенность: информация в объекте `bytes` представляет собой последовательность байтов (`byte string`), в то время как обычная строка — это последовательность символов (`character string`). Компьютер понимает первый тип, мы (люди) — второй.

Для того чтобы прочитать информацию из объекта `bytes`, ее нужно декодировать (`decode`). Если мы захотим вернуть ее обратно в объект `bytes`, соответственно, закодировать (`encode`).



Таким образом, чтобы прочитать данные напрямую из словаря **uploaded**, вначале нам нужно преобразовать эти данные в обычную строку.

```
1 # обратимся к ключу словаря uploaded и применим метод
2 .decode()
3
4 uploaded_str = uploaded['test.csv'].decode()
5
6 # на выходе получаем обычную строку
7
8 print(type(uploaded_str))
9
10
11 <class 'str'>
```

Выведем первые 35 значений.

```
1 print(uploaded_str[:35])
2
3
4 PassengerId,Pclass,Name,Sex,Age,Sib
```

Если разбить строку методом **.split()** по символам `\r` (возврат к началу строки) и `\n` (новая строка), то на выходе мы получим список.

```
1 uploaded_list = uploaded_str.split('\r\n')
2
3 type(uploaded_list)
```

```
1 list
```

Пройдемся по этому списку и выведем первые четыре значения.

```
1 # не забудем создать индекс с помощью функции  
enumerate()
```

```
2  
3 for i, line in enumerate(uploaded_list):
```

```
4  
5     # начнем выводить записи
```

```
6  
7     print(line)
```

```
8  
9     # когда дойдем до четвертой строки
```

```
10  
11     if i == 3:
```

```
12  
13         # прервемся
```

```
14  
15         break
```

```
1 PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare  
2 ,Cabin,Embarked
```

```
3 892,3,"Kelly, Mr. James",male,34.5,0,0,330911,7.8292,,Q
```

```
4 893,3,"Wilkes, Mrs. James (Ellen  
Needs)",female,47,1,0,363272,7,,S
```

```
894,2,"Myles, Mr. Thomas  
Francis",male,62,0,0,240276,9.6875,,Q
```



Вот нам и пригодился **оператор break**. Как мы видим, первая строка — это заголовок (header), остальные — информация по каждому из пассажиров.

## Использование функции `open()` и конструкции `with open()`

Такого же результата можно добиться с помощью базовой **функции `open()`**.

Обратите внимание, здесь мы читаем файл непосредственно из папки **`/content/`**. Декодировать файл уже не нужно.

Функция **`open()`** возвращает объект, который используется для чтения и изменения файла. Откроем файл `train.csv`.

```
1 # передадим функции open() адрес файла
2 # параметр 'r' означает, что мы хотим прочитать (read)
  файл
3 f1 = open('/content/train.csv', 'r')
```

Вначале попробуем применить метод **`.read()`**.

```

1 # метод .read() помещает весь файл в одну строку
2
3 # выведем первые 142 символа (если параметр не
4   указывает, выведется все содержимое)
5
6 print(f1.read(142))
7
8
9
10
11 # в конце файл необходимо закрыть
12
13 f1.close()

```

```

1 PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ti
2 cket,Fare,Cabin,Embarked
3
4 1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5
5 21171,7.25,,S
6
7
8
9

```

Для наших целей метод **.read()** не очень удобен. Будет лучше пройтись по файлу в цикле **for**.

```

1 # снова откроем файл
2
3 f2 = open('/content/train.csv', 'r')
4
5
6
7
8 # пройдемся по нашему объекту в цикле for и
9   параллельно создадим индекс
10
11 for i, line in enumerate(f2):
12
13
14
15
16
17
18   # выведем строки без служебных символов по краям
19
20   print(line.strip())
21
22
23
24

```

1  
0  
1  
1  
1  
2  
1  
3  
1  
4  
1  
5

```
# дойдя до четвертой строки, прервемся

if i == 3:

    break

# не забудем закрыть файл

f2.close()
```

```
PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5
21171,7.25,,S
2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs
Thayer)",female,38,1,0,PC 17599,71.2833,C85,C
3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2.
3101282,7.925,,S
```

Еще один способ — использовать **конструкцию with open()**. В этом случае специально закрывать файл не нужно.

```

1 # скажем Питону: "открой файл и назови его f3"
2 with open('/content/test.csv', 'r') as f3:
3
4     # "пройди по строкам без служебных символов"
5     for i, line in enumerate(f3):
6
7         print(line.strip())
8
9         # и "прервись на четвертой строке"
10        if i == 3:
11
12            break
13
14

```

```

1 PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare
2 ,Cabin,Embarked
3
4 892,3,"Kelly, Mr. James",male,34.5,0,0,330911,7.8292,,Q
5
6 893,3,"Wilkes, Mrs. James (Ellen
7 Needs)",female,47,1,0,363272,7,,S
8
9
10 894,2,"Myles, Mr. Thomas
11 Francis",male,62,0,0,240276,9.6875,,Q
12
13

```

## Чтение через библиотеку Pandas

Вероятно наиболее удобный и подходящий для наших целей способ чтения файлов — это преобразование напрямую в датафрейм библиотеки Pandas. С этим методом в целом мы уже знакомы.

```
1 # импортируем библиотеку
2
3
4 # применим функцию read_csv() и посмотрим на первые три
  записи файла train.csv
5
6 train = pd.read_csv('/content/train.csv')
7
8 train.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0

```
1 # сделаем то же самое с файлом test.csv
2
3 test = pd.read_csv('/content/test.csv')
4
5 test.head(3)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch
0	892	3	Kelly, Mr. James	male	34.5	0	0
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0

Примечание. На скриншотах приведена лишь часть датафреймов, полностью их можно посмотреть в [ноутбуке](#).

## Этап 3. Построение модели и прогноз

Давайте ненадолго отвлечемся от работы с файлами и построим несложную модель, которая предскажет погиб пассажир (обозначим этот факт через 0) или выжил (1). Прежде всего, концептуально обсудим, что нам нужно сделать.

Для понимания дальнейшей работы очень советую пройти или повторить первые три раздела [вводного курса](#). На этом занятии я предполагаю, что вы с ними уже знакомы.

- **Шаг 1.** Обработать и проанализировать данные
- **Шаг 2.** Разделить обучающую выборку (train) на признаки ( $X_{train}$ ) и целевую переменную ( $y_{train}$ )
- **Шаг 3.** Обучить модель логистической регрессии
- **Шаг 4.** Подготовить тестовые данные ( $X_{test}$ ) и построить прогноз

А теперь обо всем по порядку.

### Шаг 1. Обработка и анализ данных

#### Исследовательский анализ данных (EDA)

Напомню, что основная задача EDA — выявить взаимосвязь между признаками и целевой переменной. Воспользуемся методом `.info()`, чтобы обобщенно посмотреть на наши данные.

```
1 train.info()
```

```
1 <class 'pandas.core.frame.DataFrame'>
```

```
2 RangeIndex: 891 entries, 0 to 890
```

```
3 Data columns (total 12 columns):
```

```
4 #    Column    Non-Null Count  Dtype
```

```
5 ---  -
```

```
6 0    PassengerId  891 non-null    int64
```

```
7 1    Survived    891 non-null    int64
```

```
8 2    Pclass      891 non-null    int64
```

```
9 3    Name        891 non-null    object
```

```
1 4    Sex         891 non-null    object
```

```
0 5    Age         714 non-null    float64
```

```
1 6    SibSp       891 non-null    int64
```

```
1 7    Parch       891 non-null    int64
```

```
2 8    Ticket      891 non-null    object
```

```
1 9    Fare        891 non-null    float64
```

```
3 10   Cabin        204 non-null    object
```

```
11  Embarked  889 non-null    object
```

```
1 dtypes: float64(2), int64(5), object(5)
4 memory usage: 83.7+ KB
```

Как мы видим, у нас 12 переменных. Одна из них (Survived) — зависимая (целевая), остальные — независимые (признаки). Всего в датасете 891 запись, при этом в нескольких переменных есть пропуски.

Ниже приведено короткое описание каждой из переменных:

- PassengerId — id пассажира
- Survived — погиб (0) или выжил (1)
- Pclass — класс билета (первый (1), второй (2) или третий (3))
- Name — имя пассажира
- Sex — пол
- Age — возраст
- SibSp — количество братьев и сестер или супругов на борту
- Parch — количество родителей и детей на борту
- Ticket — номер билета



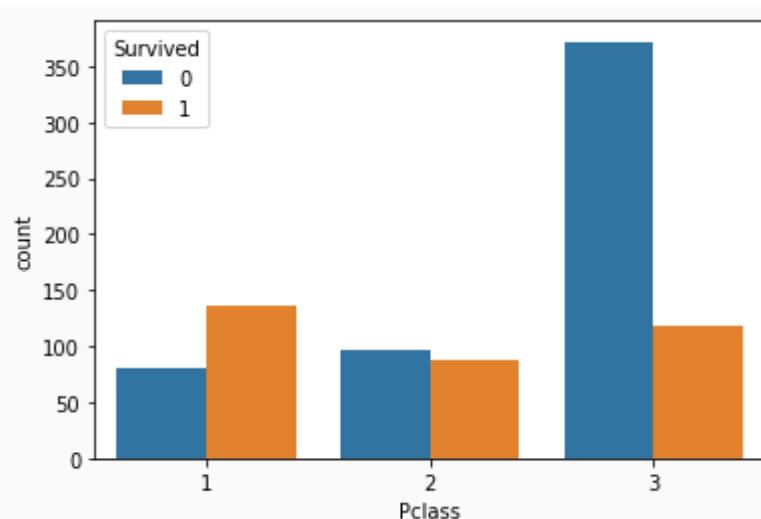
- Fare — стоимость билета
- Cabin — номер каюты
- Embarked — порт посадки (C — Шербур; Q — Квинстаун; S — Саутгемптон)

Проведем несложный визуальный анализ данных.

```
1 # для построения графиков воспользуемся новой для нас
  библиотекой seaborn
2
  import seaborn as sns
```

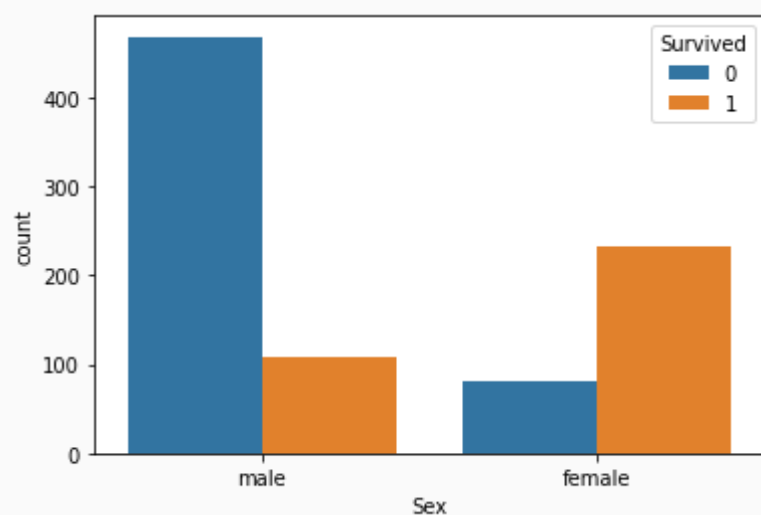
У нас есть несколько потенциально значимых категориальных переменных, целевая переменная — тоже категориальная. В этом случае удобно использовать [столбчатую диаграмму](#) (bar chart), где каждый столбец также разбит на категории. В библиотеке seaborn такую диаграмму можно построить с помощью функции `countplot()`.

```
1 # посмотрим насколько значим класс билета для
  выживания пассажира
2
  # с помощью x и hue мы можем уместить две
3 категориальные переменные на одном графике
  sns.countplot(x = 'Pclass', hue = 'Survived', data =
    train)
```



Мы видим, что погибших пассажиров в третьем классе гораздо больше, чем выживших. При этом в первом классе больше выживших, чем погибших. Очевидно класс билета имеет значение.

```
1 # кто выживал чаще, мужчины или женщины?  
2 sns.countplot(x = 'Sex', hue = 'Survived', data =  
train)
```



Большинство мужчик погибло. Большая часть женщин выжила. Пол также значим для построения прогноза.

## Пропущенные значения

Посмотрим, что можно сделать с пропущенными значениями (missing values).

```
1 # выявим пропущенные значения с помощью .isnull() и
2 посчитаем их количество sum()
3
4 train.isnull().sum()
```

```
1 PassengerId      0
2 Survived         0
3 Pclass           0
4 Name             0
5 Sex              0
6 Age             177
7 SibSp            0
8 Parch            0
9 Ticket           0
10 Fare            0
11 Cabin           687
12 Embarked        2
13
14 dtype: int64
15
16
```

1  
3

Больше всего пропущенных значений в переменной Cabin. Они также есть в переменных Age и Embarked.

```
1 # переменная Cabin (номер каюты), скорее всего, не  
# является самой важной  
2  
3 # избавимся от нее с помощью метода .drop()  
4 # (параметр axis отвечает за столбцы, inplace = True  
# сохраняет изменения)  
  
train.drop(columns = 'Cabin', axis = 1, inplace =  
True)
```

```
1 # а вот Age (возраст) точно важен, заменим пустые  
# значения средним арифметическим  
2  
train['Age'].fillna(train['Age'].mean(), inplace =  
True)
```

В данном случае мы применили метод **.fillna()**, то есть «заполнить пропуски», к столбцу Age (через `train['Age']`) и заполнили пропуски средним значением этого же столбца через `train['Age'].mean()`.

Более подробно с преобразованием датафреймов мы познакомимся на [курсе анализа и обработки данных](#). На данном этапе важно просто понимать логику нашей работы.

```
1 # у нас остаются две пустые строки в Embarked, удалим их
2
train.dropna(inplace = True)
```

Посмотрим на результат.

```
1 train.isnull().sum()
```

```
1 PassengerId    0
```

```
2 Survived      0
```

```
3 Pclass        0
```

```
4 Name          0
```

```
5 Sex           0
```

```
6 Age           0
```

```
7 SibSp         0
```

```
8 Parch         0
```

```
9 Ticket        0
```

```
1 Fare          0
```

```
0 Embarked      0
```

```
1 dtype: int64
```

```
1
```

```
1
```

```
2
```

## Категориальные переменные

Теперь нужно поработать с категориальными переменными (categorical variable). Как мы помним, модель не сможет подобрать веса, если значения выражены словами (например, male и female в переменной Sex или C, Q, S в переменной Embarked).

Кроме того, когда категория выражена не 0 и 1, мы все равно не можем оставить ее без изменения. Например, если не трогать переменную Pclass, то модель воспримет классы 1, 2 и 3 как количественную переменную (проще говоря, число), а не как категорию.

И в первом, и во втором случае к переменным нужно применить one-hot encoding. Мы уже познакомились с этим методом, когда разбирали [нейронные сети](#). В библиотеке Pandas есть метод .get\_dummies(), который как раз и выполнит необходимые преобразования.

В статистике, dummy variable или вспомогательная переменная — это переменная, которая принимает значения 0 или 1 в зависимости от наличия определенного признака.

Применим этот метод на практике.

```
1 | # применим one-hot encoding к переменной Sex (пол) с  
   | помощью метода .get_dummies()
```

```
2 pd.get_dummies(train['Sex']).head(3)
```

	female	male
0	0	1
1	1	0
2	1	0

Первый пассажир — мужчина (в колонке male стоит 1), второй и третий — женщина. Помимо этого, если присмотреться, то станет очевидно, что мы можем обойтись только одним столбцом. В частности, в столбце male уже содержится достаточно информации о поле (если 1 — мужчина, если 0 — женщина). Это значит, что первый столбец можно удалить.

```
1 # удалим первый столбец, он избыточен
2 sex = pd.get_dummies(train['Sex'], drop_first = True)
3 sex.head(3)
```

	male
0	1
1	0
2	0

Сделаем то же самое для переменных Pclass и Embarked.

```
1 embarked = pd.get_dummies(train['Embarked'],
2 drop_first = True)
3 pclass = pd.get_dummies(train['Pclass'], drop_first =
4 True)
```

Еще раз замечу, что переменную Survived трогать не надо. Она уже выражена через 0 и 1.

Присоединим новые (закодированные) переменные к исходному датафрейму train. Для этого используем функцию .concat().

```
1 train = pd.concat([train, pclass, sex, embarked],  
axis = 1)
```

## Отбор признаков

Теперь давайте отберем те переменные (feature selection), которые мы будем использовать в модели.

- В первую очередь, удалим исходные (до применения one-hot encoding) переменные Sex, Pclass и Embarked
- Кроме того, переменные PassengerId, Name и Ticket вряд ли скажут что-то определенное о шансах на выживание пассажира, удалим и их

```
1 # применим функцию drop() к соответствующим столбцам  
2 train.drop(['PassengerId', 'Pclass', 'Name', 'Sex',  
3 'Ticket', 'Embarked'], axis = 1, inplace = True)  
train.head(3)
```



	Survived	Age	SibSp	Parch	Fare	2	3	male	Q	S
0	0	22.0	1	0	7.2500	0	1	1	0	1
1	1	38.0	1	0	71.2833	0	0	0	0	0
2	1	26.0	0	0	7.9250	0	1	0	0	1

Как вы видите, теперь все переменные либо количественные (Age, SibSp, Parch, Fare), либо категориальные и выражены через 0 и 1.

## Нормализация данных

На занятиях по [классификации](#) и [кластеризации](#) мы уже говорили о важности приведения количественных переменных к одному масштабу. В противном случае модель может неоправданно придать большее значение признаку с большим масштабом.

```

1  # импортируем класс StandardScaler
2
3  from sklearn.preprocessing import StandardScaler
4
5  # создадим объект этого класса
6
7  scaler = StandardScaler()
8
9  # выберем те столбцы, которые мы хотим масштабировать
10
11 cols_to_scale = ['Age', 'Fare']
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

1
0 # применим их
1
1 train[cols_to_scale] =
1 scaler.transform(train[cols_to_scale])
1
2
1 # посмотрим на результат
1
3 train.head(3)
1
4
1
5
1
6
1
7

```

	Survived	Age	SibSp	Parch	Fare	2	3	male	Q	S
0	0	-0.590495	1	0	-0.500240	0	1	1	0	1
1	1	0.643971	1	0	0.788947	0	0	0	0	0
2	1	-0.281878	0	0	-0.486650	0	1	0	0	1

Остается небольшой технический момент. Переменные 2 и 3 (второй и третий класс) выражены числами, а не строками (их выдает отсутствие кавычек в коде ниже). Так быть не должно.

```

1 train.columns

```

```
1 Index(['Survived', 'Age', 'SibSp', 'Parch', 'Fare', 2,  
3, 'male', 'Q', 'S'], dtype='object')
```

Преобразуем эти переменные в тип `str` через функцию `map()`.

```
1 train.columns = train.columns.map(str)
```

## Шаг 2. Разделение обучающей выборки на признаки и целевую переменную

```
1 # поместим в X_train все кроме столбца Survived  
2 X_train = train.drop('Survived', axis = 1)  
3  
4 # столбец 'Survived' станет нашей целевой переменной  
   (y_train)  
5 y_train = train['Survived']
```

```
1 X_train.head(3)
```

	Age	SibSp	Parch	Fare	2	3	male	Q	S
0	-0.590495	1	0	-0.500240	0	1	1	0	1
1	0.643971	1	0	0.788947	0	0	0	0	0
2	-0.281878	0	0	-0.486650	0	1	0	0	1

## Шаг 3. Обучение модели логистической регрессии

Воспользуемся **моделью логистической регрессии** из библиотеки `sklearn` и передадим ей обучающую выборку.

```
1 # импортируем логистическую регрессию из модуля
   linear_model библиотеки sklearn
2
3 from sklearn.linear_model import LogisticRegression
4
5 # создадим объект этого класса и запишем его в
   переменную model
6 model = LogisticRegression()
7
8 # обучим нашу модель
   model.fit(X_train, y_train)
9
10 LogisticRegression()
```

Остается сделать прогноз и оценить качество модели. При этом обратите внимание, что в тестовых данных отсутствует целевая переменная (почему это так, расскажу ниже), поэтому чтобы иметь хоть какое-то представление о качестве модели, нам необходимо вначале использовать обучающую выборку для построения **прогноза**.

```
1 # сделаем предсказание класса на обучающей выборке
2 y_pred_train = model.predict(X_train)
```

Теперь мы можем сравнить прогнозные значения с фактическими.  
Построим **матрицу ошибок** (confusion matrix).

```
1 # построим матрицу ошибок
2 from sklearn.metrics import confusion_matrix
3
4 # передадим ей фактические и прогнозные значения
5 conf_matrix = confusion_matrix(y_train, y_pred_train)
6
7 # преобразуем в датафрейм
8 conf_matrix_df = pd.DataFrame(conf_matrix)
9 conf_matrix_df
```

	0	1
0	478	71
1	103	237

Для удобства интерпретации добавим подписи.

```
conf_matrix_labels = pd.DataFrame(conf_matrix, columns =
['Прогноз погиб', 'Прогноз выжил'], index = ['Факт
погиб', 'Факт выжил'])

conf_matrix_labels
```

	Прогноз погиб	Прогноз выжил
Факт погиб	478	71
Факт выжил	103	237

Также давайте посмотрим на [метрику accuracy](#). Она показывает долю правильно предсказанных значений. То есть мы берем тех, кого верно предсказали как погибших (true negative, TN, таких было 478), и тех, кого верно предсказали как выживших (true positive, TP, 237), и делим на общее число прогнозов.

```
1 # рассчитаем метрику accuracy вручную
2 round((478 + 237)/(478 + 237 + 71 + 103), 3)

1 0.804

1 # импортируем метрику accuracy из sklearn
2 from sklearn.metrics import accuracy_score
3
4 # так же передадим ей фактические и прогнозные значения
5 model_accuracy = accuracy_score(y_train, y_pred_train)
6
7 # округлим до трех знаков после запятой
8 round(model_accuracy, 3)
```

```
1 0.804
```

На обучающей выборке наша модель показала результат в 80,4%. При этом **только на тестовой выборке** мы можем объективно оценить качество нашего алгоритма.

## Шаг 4. Построение прогноза на тестовых данных

Посмотрим на тестовые данные.

```
1 test.info()
```

```
1 <class 'pandas.core.frame.DataFrame'>
```

```
2 RangeIndex: 418 entries, 0 to 417
```

```
3 Data columns (total 11 columns):
```

```
4 #      Column      Non-Null Count  Dtype
```

```
5 ---  -
```

```
6 0  PassengerId  418 non-null    int64
```

```
7 1  Pclass      418 non-null    int64
```

```
8 2  Name        418 non-null    object
```

```
9 3  Sex          418 non-null    object
```

```
4  Age          332 non-null    float64
```

```
1 5 SibSp      418 non-null   int64
0 6 Parch      418 non-null   int64
1 7 Ticket     418 non-null   object
1
1 8 Fare       417 non-null   float64
1
2 9 Cabin      91 non-null     object
1
1 10 Embarked  418 non-null   object
3
dtypes: float64(2), int64(4), object(5)
1
memory usage: 36.0+ KB
4
```

Взглянув на сводку по тестовым данным, становится заметна одна сложность. Мы обучили модель на *обработанных* данных. В частности, мы заполнили пропуски, закодировали категориальные переменные и убрали лишние признаки. Кроме того, мы масштабировали количественные переменные и превратили названия столбцов в строки.

Для того чтобы наша модель смогла работать с тестовой выборкой нам нужно таким же образом обработать и эти данные.



При этом обратите внимание, мы не нарушаем принципа разделения данных, поскольку меняем тестовую выборку *так же*, как мы меняли обучающую.

```
1 # для начала дадим датасету привычное название X_test
```

```
2 X_test = test
```

```
1 # заполним пропуски в переменных Age и Fare средним  
арифметическим
```

```
2 X_test['Age'].fillna(test['Age'].mean(), inplace =  
3 True)
```

```
X_test['Fare'].fillna(test['Fare'].mean(), inplace =  
True)
```

```
1 # выполним one-hot encoding категориальных переменных
```

```
2 sex = pd.get_dummies(X_test['Sex'], drop_first =  
True)
```

```
3 embarked = pd.get_dummies(X_test['Embarked'],  
4 drop_first = True)
```

```
pclass = pd.get_dummies(X_test['Pclass'], drop_first  
= True)
```

```

1 # присоединим новые столбцы к исходному датафрейму
2 X_test = pd.concat([test, pclass, sex, embarked],
3                     axis = 1)
4
5 # и удалим данные, которые теперь не нужны
6 X_test.drop(['PassengerId', 'Pclass', 'Name', 'Sex',
7             'Cabin', 'Ticket', 'Embarked'], axis = 1, inplace =
8             True)
9
10 # посмотрим на результат
11 X_test.head(3)

```

	Age	SibSp	Parch	Fare	2	3	male	Q	S
0	34.5	0	0	7.8292	0	1	1	1	0
1	47.0	1	0	7.0000	0	1	0	0	1
2	62.0	0	0	9.6875	1	0	1	1	0

Теперь нужно масштабировать количественные переменные. Для этого мы будем использовать те параметры (среднее арифметическое и СКО), которые мы получили при обработке обучающей выборки. Так мы сохраним единообразие изменений и избежим [утечки данных](#) (data leakage).

```

1 # применим среднее арифметическое и СКО обучающей
2   выборки для масштабирования тестовых данных
3
4 X_test[cols_to_scale] =
5 scaler.transform(X_test[cols_to_scale])

```

```
X_test.head(3)
```

	Age	SibSp	Parch	Fare	2	3	male	Q	S
0	0.373932	0	0	-0.488579	0	1	1	1	0
1	1.338358	1	0	-0.505273	0	1	0	0	1
2	2.495670	0	0	-0.451165	1	0	1	1	0

Остается превратить название столбцов в строки.

```
1 X_test.columns = X_test.columns.map(str)
```

И сделать прогноз на тестовой выборке.

```
1 y_pred_test = model.predict(X_test)
```

На выходе мы получаем массив с прогнозами.

```
1 # посмотрим на первые 10 прогнозных значений
```

```
2 y_pred_test[:10]
```

```
1 array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0])
```

## Этап 4. Сохранение результата в новом файле на сервере

Теперь, когда прогноз готов, мы можем сформировать новый файл, назовем его `result.csv`, в котором будет содержаться `id` пассажира и результат, погиб или нет. Приведу **пример** того, что мы хотим получить.

```
1 # файл с примером можно загрузить не с локального
  компьютера, а из Интернета
2
3 url =
4 'https://www.dmitrymakarov.ru/wp-content/uploads/2021
  /11/titanic_example.csv'
5
6 # просто поместим его url в функцию read_csv()
  example = pd.read_csv(url)
  example.head(3)
```

	PassengerId	Survived
0	892	0
1	893	1
2	894	0

Перед **созданием** нужного нам файла (1) соберем данные в новый датафрейм.

```

1 # возьмем индекс пассажиров из столбца PassengerId
  тестовой выборки
2
3 ids = test['PassengerId']
4
5 # создадим датафрейм из словаря, в котором
6 # первая пара ключа и значения - это id пассажира,
  вторая - прогноз "на тесте"
7
8 result = pd.DataFrame({'PassengerId': ids,
  'Survived': y_pred_test})
9
10 # посмотрим, что получилось
11
12 result.head()

```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1

И (2) создадим новый файл.

```

1 # создадим новый файл result.csv с помощью функции
  to_csv(), удалив при этом индекс
2
3 result.to_csv('result.csv', index = False)

```

```

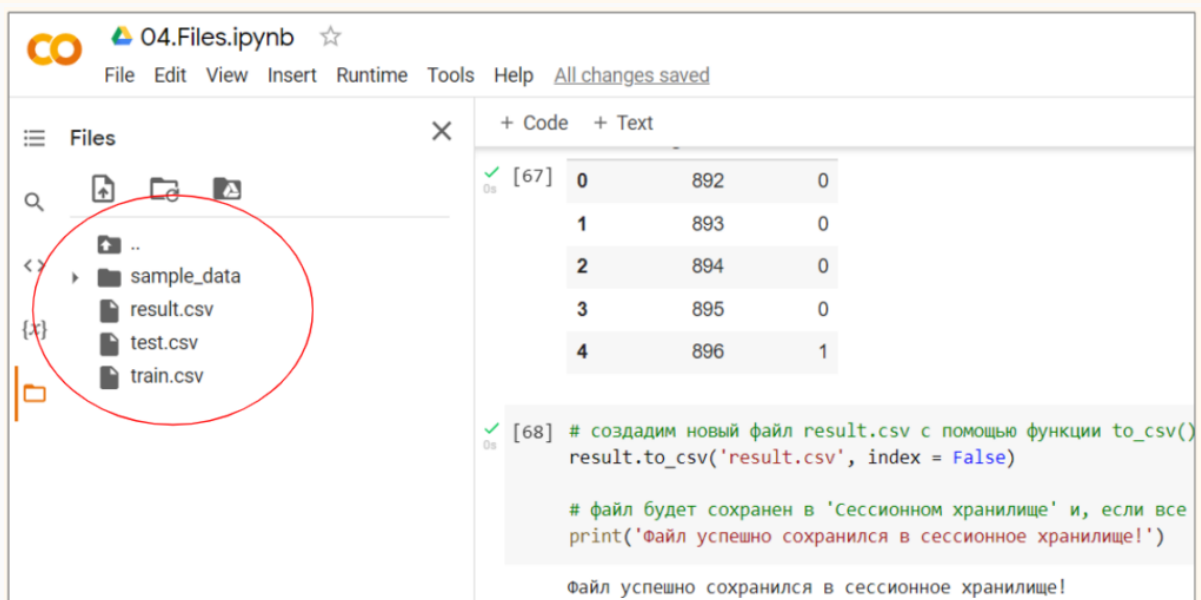
4 # файл будет сохранен в 'Сессионном хранилище' и,
5 если все пройдет успешно, выведем следующий текст:

print('Файл успешно сохранился в сессионное
хранилище!')
```

```

1 Файл успешно сохранился в сессионное хранилище!
```

Новый файл появится в «Сессионном хранилище».



The screenshot shows a Jupyter Notebook titled '04.Files.ipynb'. On the left, the 'Files' pane displays a directory structure: a folder named 'sample\_data' and three files: 'result.csv', 'test.csv', and 'train.csv'. The 'result.csv' file is circled in red. On the right, the code cell shows the execution of a file saving operation. The output is a table with 4 rows and 3 columns, followed by a confirmation message.

0	892	0
1	893	0
2	894	0
3	895	0
4	896	1

Below the table, the code cell shows the following text:

```

[68] # создадим новый файл result.csv с помощью функции to_csv()
result.to_csv('result.csv', index = False)

# файл будет сохранен в 'Сессионном хранилище' и, если все
print('Файл успешно сохранился в сессионное хранилище!')
```

At the bottom of the code cell, the output message reads: 'Файл успешно сохранился в сессионное хранилище!'

## Этап 5. Скачивание обратно на жесткий диск

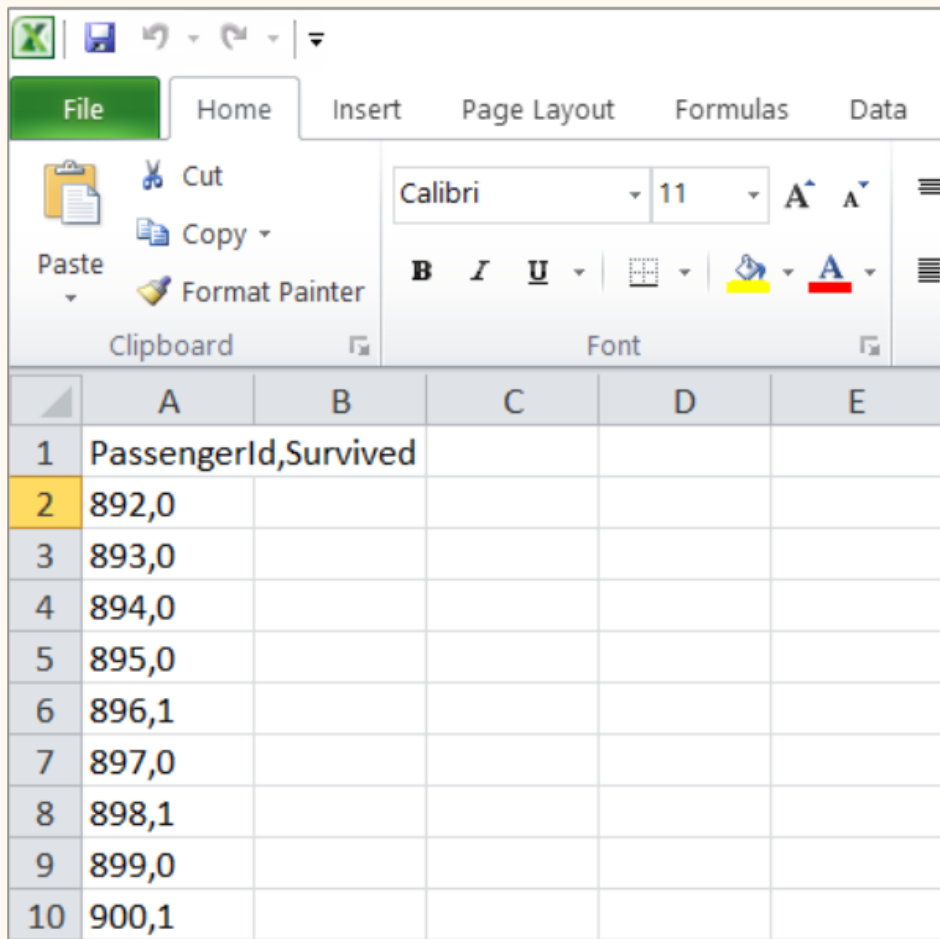
После этого мы можем скачать файл на жесткий диск.

```

1 # применим метод .download() объекта files

files.download('/content/result.csv')
```

Вот что у нас получилось.



	A	B	C	D	E
1	PassengerId,Survived				
2	892,0				
3	893,0				
4	894,0				
5	895,0				
6	896,1				
7	897,0				
8	898,1				
9	899,0				
10	900,1				

## Связывание проекта с GitHub

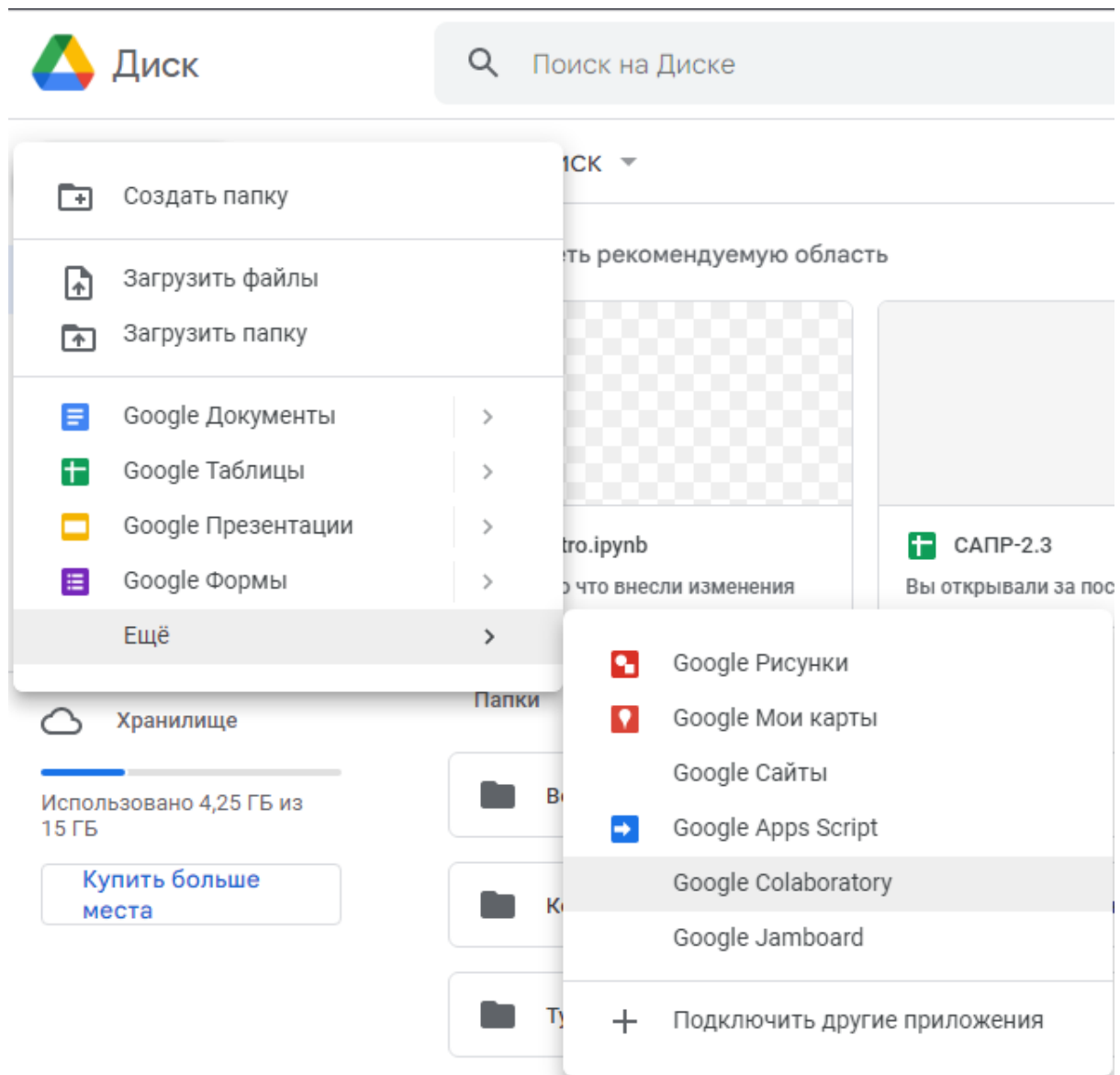
###Что такое ветвление?

Удобная поддержка ветвления — важное свойство Git. Использование ветвления позволяет решать отдельные задачи, не вмешиваясь в основную линию разработки.

Ветка в Git — это последовательность коммитов. С технической точки зрения ветка — это указатель или ссылка на последний коммит в этой ветке. По умолчанию, имя

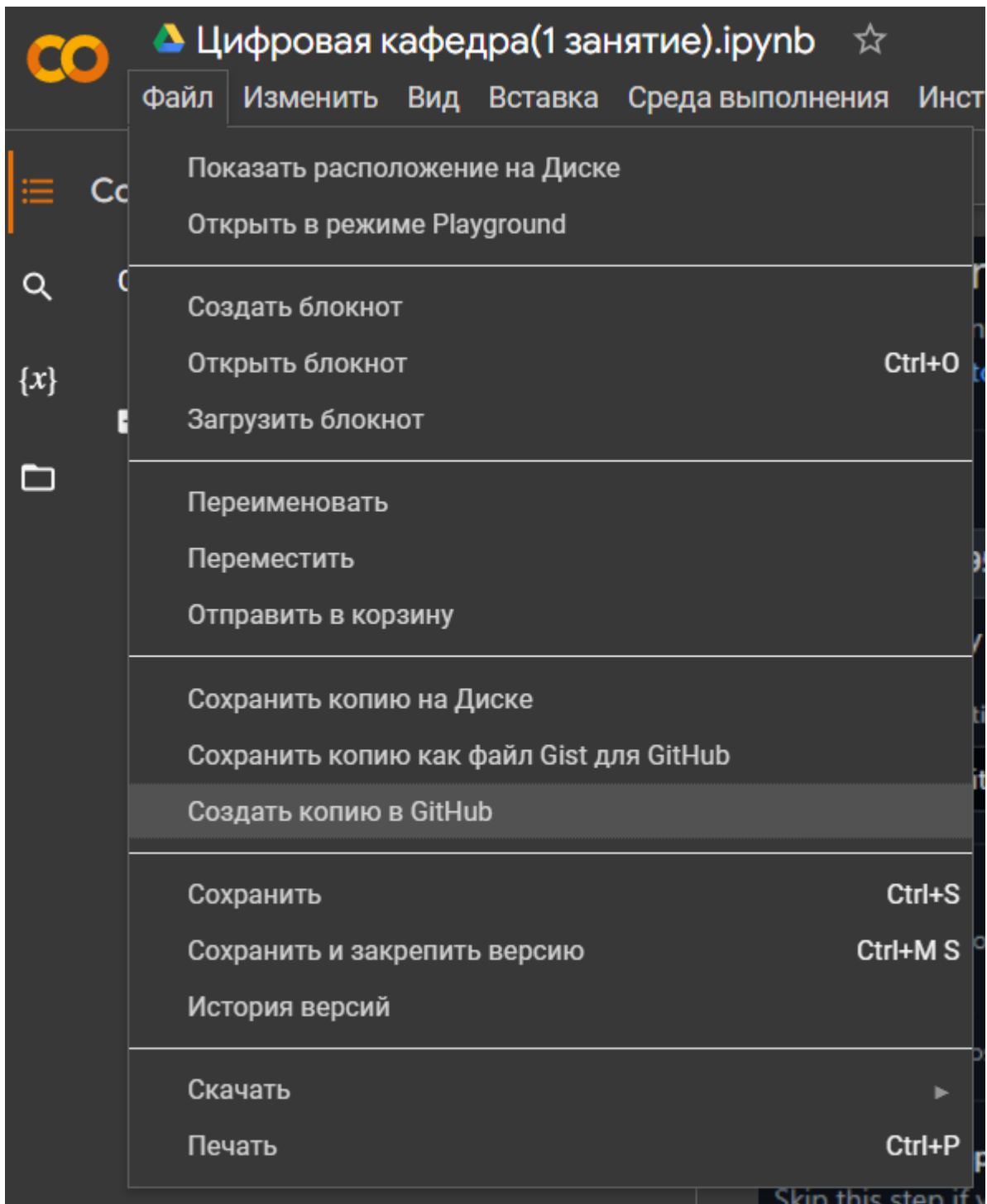
основной ветки в Git — master. Каждый раз, когда создается новый коммит, указатель ветки master автоматически передвигается на него.

При создании новой ветки коммиту дается новый указатель, например testing. Если переключиться на ветку testing и сделать новый коммит, то указатель на ветку testing переместится вперед, тогда как указатель на основную ветку master останется на месте. Переключившись обратно на ветку master, файлы в рабочем каталоге вернутся в состояние коммита, на который указывает master.




Созданный файл можно связать с GitHub





## Копирование в GitHub

Хранилище: 

fokro1995/collab



Ветвь: 

main



Путь к файлу

Цифровая\_кафедра(1\_занятие).ipynb

Сообщение к фиксации

Создано с помощью Colaboratory

☒ Добавлять ссылку на Colaboratory


Отмена

OK

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*

 fokro1995 ▾

Repository name \*

/ collab ✓

Great repository names are short and memorable. Need inspiration? How about [scaling-winner?](#)

Description (optional)

Testing repository for working with Google collab

☒



**Public**

Anyone on the internet can see this repository. You choose who can commit.

☐



**Private**

You choose who can see and commit to this repository.

**Initialize this repository with:**

Skip this step if you're importing an existing repository.

☒ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

**Add .gitignore**


Choose which files not to track from a list of templates. [Learn more.](#)


.gitignore template: Python ▾

**Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a private repository in your personal account.

Create repository

---

После этого можно выбрать подходящий ускоритель под имеющийся проект

---

Среда выполнения	Инструменты	Справка	Изменения сохранены
Выполнить все			Ctrl+F9
Выполнить до этой			Ctrl+F8
Выполнить код в сфокусированной ячейке			Ctrl+Enter
Запустить выбранный код			Ctrl+Shift+Enter
Выполнить ниже			Ctrl+F10
Прервать выполнение кода			Ctrl+M I
Перезапустить среду выполнения			Ctrl+M .
Перезапустить и выполнить все			
Отключиться от среды выполнения и удалить ее			
Сменить среду выполнения			
Управление сеансами			
Показать журналы среды выполнения			