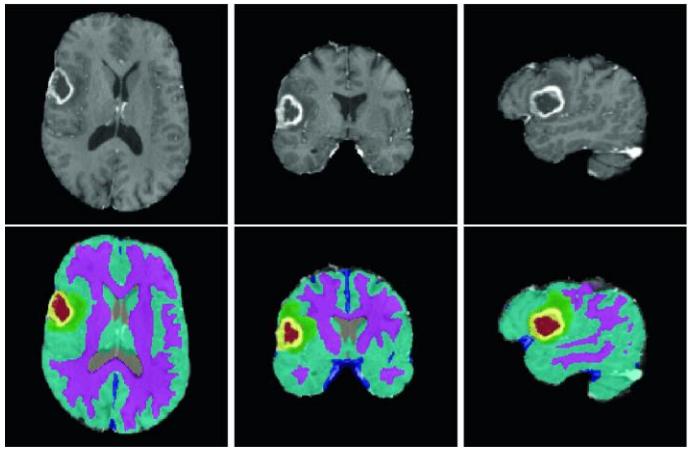
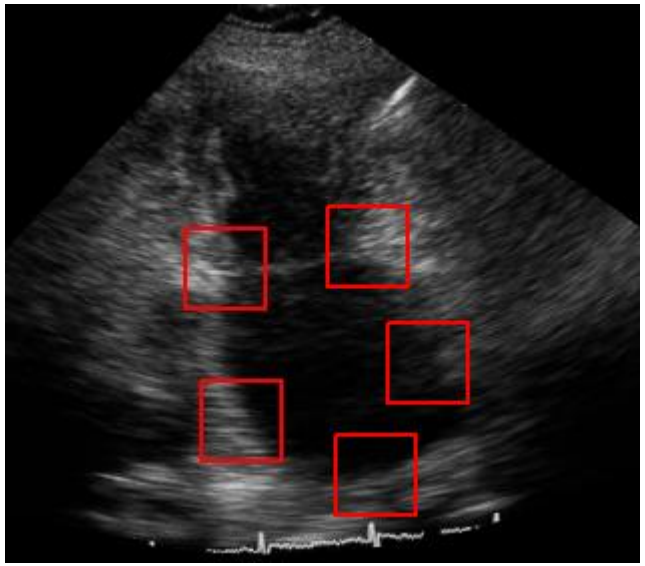
# Лабораторная работа 2. Использование открытых баз данных для распознавания медицинских изображений





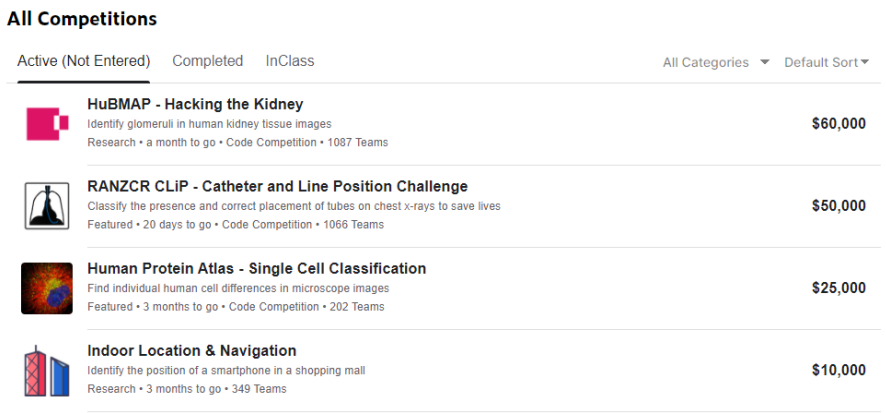
Цели работы:

1. Ознакомиться с площадкой для проведения соревнований по машинному обучению Kaggle.

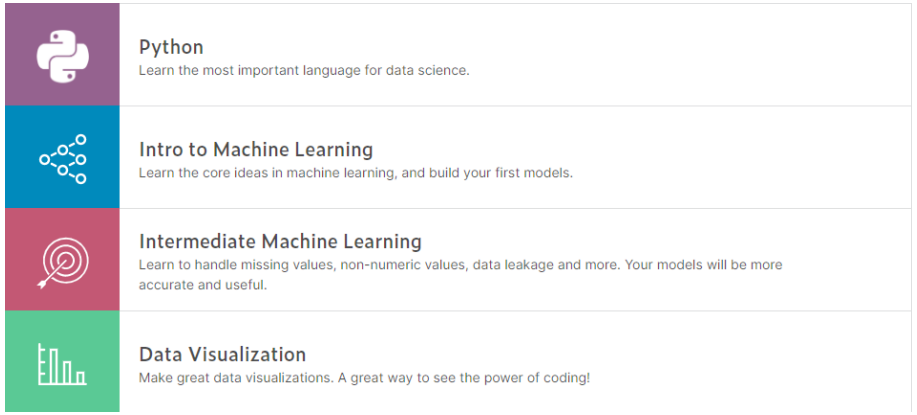
2. Ознакомиться с инструментами языка Python 3 для задач машинного обучения.

# Что такое Kaggle?

## Соревнования



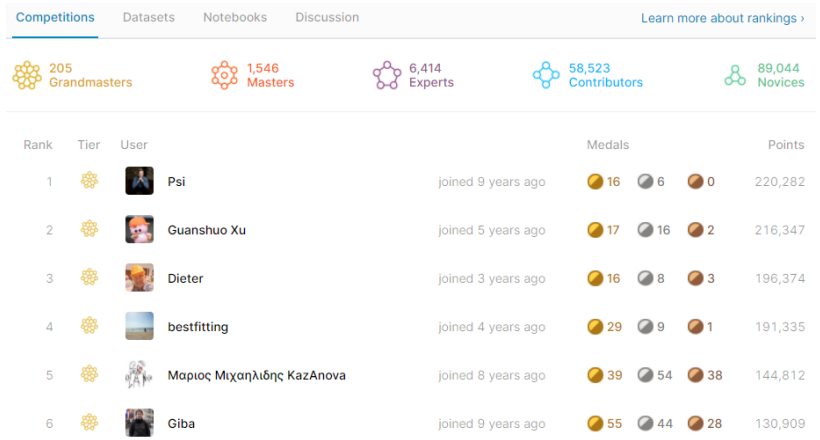

## Курсы




Почему стоит этим заняться:

- Получение опыта
- Можно использовать в резюме
- Большое сообщество людей у которых есть чему научиться

## Способ заявить о себе

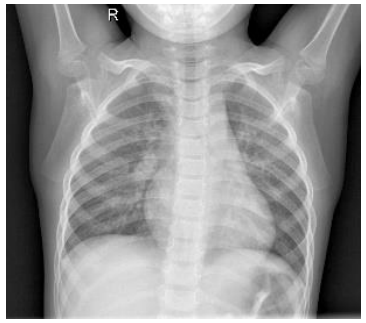



[Kaggle: Your Home for Data Science]

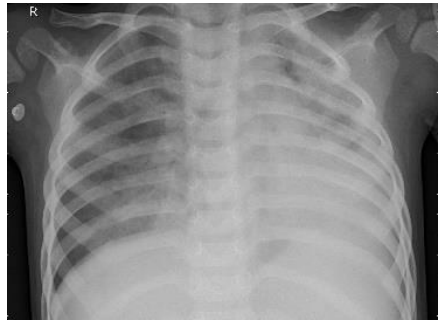
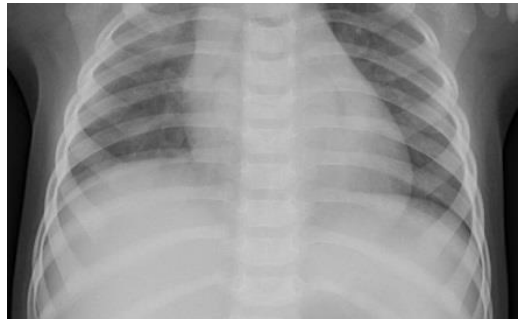
# Датасет

Набор рентгеновских снимков грудной клетки 2 классов:

Норма



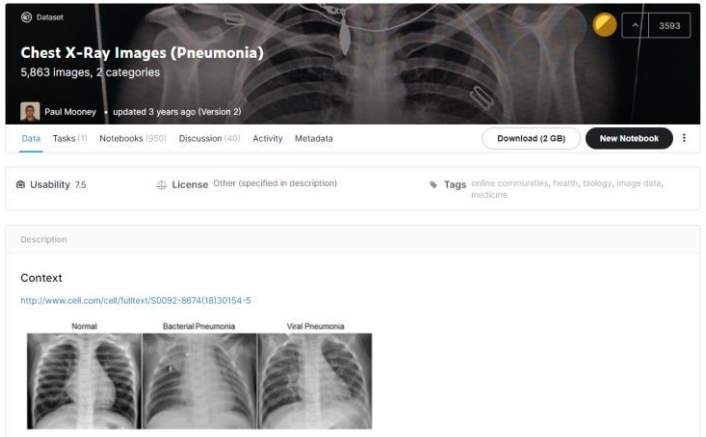

Пневмония

Снимки распределены по трем папкам:

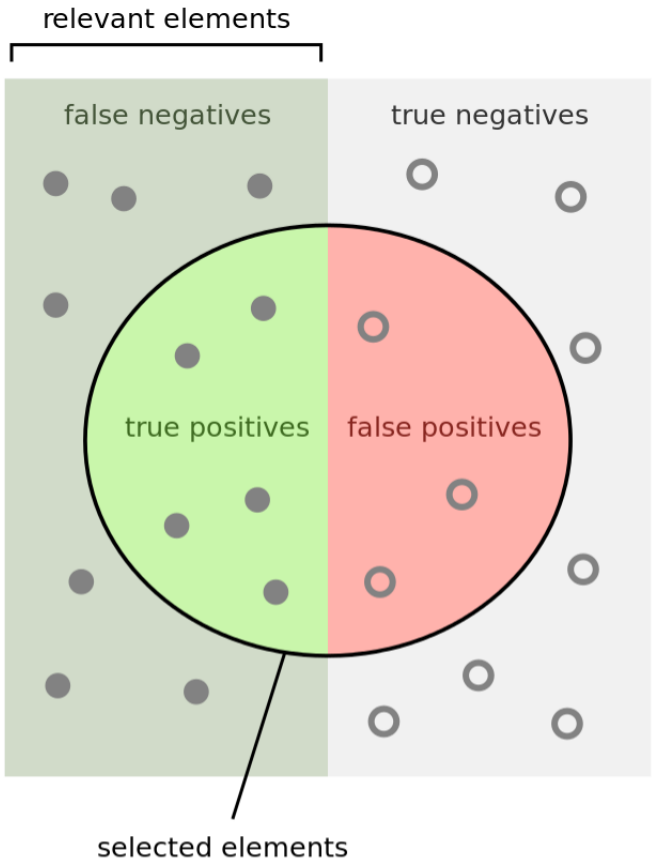
Train: 1341 снимков – норма, 3875 – пневмония.
Val: 8 снимков – норма, 8 – пневмония.
Test: 234 снимков – норма, 390 – пневмония.

- Все рентгеновские снимки грудной клетки выполнялись как часть обычного клинического ухода за пациентами
- Низкокачественные и нечитаемые снимки были исключены
- Диагнозы были поставлены двумя опытными врачами
- Для исключения ошибок полученный результаты были проверены третьим экспертом




[Chest X-Ray Images (Pneumonia) | Kaggle]

# Метрика для несбалансированных наборов данных




$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

**Precision**: Какая часть положительных предсказаний была на самом деле правильной?

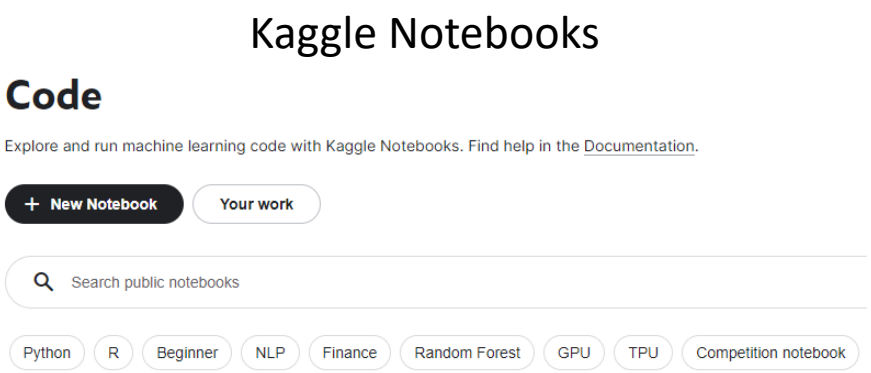
**Recall**: Какая доля реальных положительных примеров была определена правильно?

**F1**: Использует Precision и Recall для общей оценки.

[Evaluating ML Models: Precision, Recall, F1]

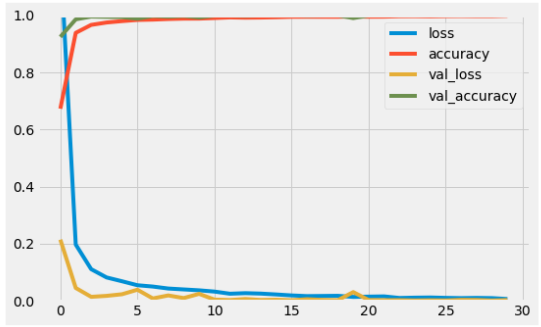
# Где работать с данными, если нет мощного компьютера?

## Kaggle Notebooks




Evaluating the model

```
In [14]: pd.DataFrame(history.history).plot(figsize=(8, 5))
         plt.grid(True)
         plt.gca().set_ylim(0, 1)
         plt.show()
```



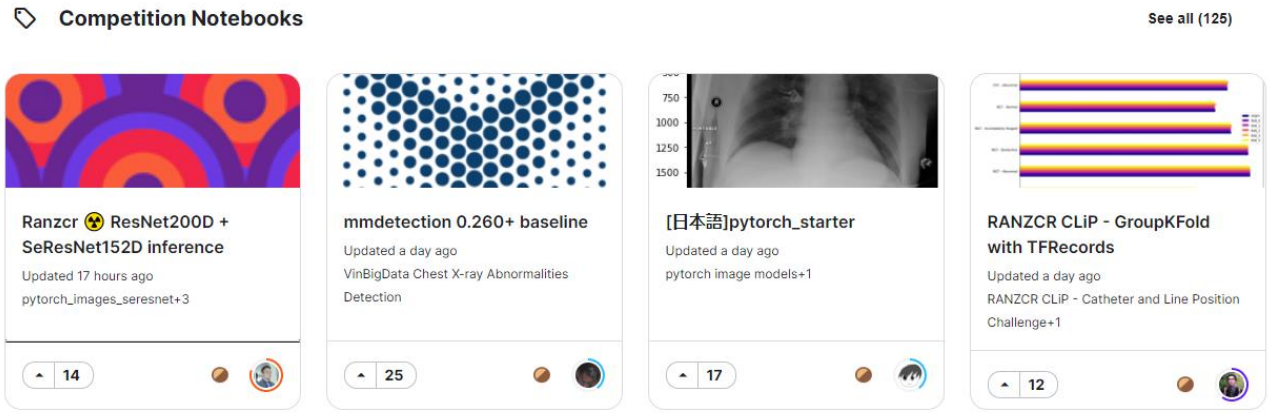

Splitting the data into train and validation set

```
In [9]: X_train, X_val, y_train, y_val = train_test_split(image_data, image_labels, test_size=0.3, rand
        om_state=42, shuffle=True)

        X_train = X_train/255
        X_val = X_val/255

        print("X_train.shape", X_train.shape)
        print("X_valid.shape", X_val.shape)
        print("y_train.shape", y_train.shape)
        print("y_valid.shape", y_val.shape)
```

```
X_train.shape (27446, 30, 30, 3)
X_valid.shape (11763, 30, 30, 3)
y_train.shape (27446,)
y_valid.shape (11763,)
```

Что можно делать?

- Писать код
- Управлять версиями
- Публиковать код
- Обсуждать свои и чужие решения
- Изучать код других людей




Kaggle Notebooks

# Язык программирования


CLASS `torch.utils.data.DataLoader(dataset: torch.utils.data.dataset.Dataset[T_co], batch_size: Optional[int] = 1, shuffle: bool = False, sampler: Optional[torch.utils.data.sampler.Sampler[int]] = None, batch_sampler: Optional[torch.utils.data.sampler.Sampler[Sequence[int]]] = None, num_workers: int = 0, collate_fn: Callable[[List[T], Any] = None, pin_memory: bool = False, drop_last: bool = False, timeout: float = 0, worker_init_fn: Callable[int, None] = None, multiprocessing_context=None, generator=None, *, prefetch_factor: int = 2, persistent_workers: bool = False)` [SOURCE]

Data loader. Combines a dataset and a sampler, and provides an iterable over the given dataset.

The `DataLoader` supports both map-style and iterable-style datasets with single- or multi-process loading, customizing loading order and optional automatic batching (collation) and memory pinning.

See `torch.utils.data` documentation page for more details.

Parameters

- **dataset** (*Dataset*) – dataset from which to load the data.
- **batch_size** (*int, optional*) – how many samples per batch to load (default: `1`).
- **shuffle** (*bool, optional*) – set to `True` to have the data reshuffled at every epoch (default: `False`).
- **sampler** (*Sampler or Iterable, optional*) – defines the strategy to draw samples from the dataset. Can be any

Почему Python?

- Быстро осваивается
- Больше количество руководств и документации
- Интерактивность
- Множество библиотек и фреймворков для машинного обучения

```
1 # Python
2 # Use Decimal to do high precision calculation
3
4 >>> 2.36 * 5.1
5 12.035999999999998
6
7 >>> from decimal import Decimal
8 >>> result = Decimal("2.36") * Decimal("5.1")
9 >>> result
10 Decimal('12.036')
11 >>> a = float(result)
12 >>> a
13 12.036
14
```


[Welcome to Python.org]

# Фреймворк машинного обучения

PyTorch

Что можно делать?

- Работать с тензорами
- Автоматически вычислять градиент
- Использовать конструктор для построения своих моделей
- Все это на GPU

```
a = torch.tensor([1,2,3,4], dtype = torch.float64)

print(a)
```

```
tensor([1., 2., 3., 4.], dtype=torch.float64)
```

```
a = torch.randn((3,3,3))

print(a)
```

```
tensor([[[ 0.8034,  3.1001,  0.1028],
         [-0.7967, -0.8711, -1.3769],
         [-1.6110,  0.5500, -1.6429]],

        [[-1.3078, -0.2058,  0.4480],
         [-0.2859, -1.5177,  0.7204],
         [ 1.3631,  0.5193, -0.6224]],

        [[ 0.6159, -1.8170,  0.3920],
         [ 1.0218, -0.6155, -0.6653],
         [-1.9467, -0.4836, -0.4640]]])
```

```
net = torch.nn.Sequential(
      torch.nn.Linear(3, 4),
      torch.nn.Sigmoid(),
      torch.nn.Linear(4, 1),
      torch.nn.Sigmoid()
      )

print(net)
```

```
Sequential(
  (0): Linear(in_features=3, out_features=4, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=4, out_features=1, bias=True)
  (3): Sigmoid()
)
```

```
a = torch.randn((2,4), dtype=torch.float32, requires_grad=True)
b = torch.randn((4,1), dtype=torch.float32, requires_grad=True)
c = a.mm(b)
d = c.sum()

print(a)
print(b)
print(c)
print(d)
```

```
tensor([[ 1.3776,  0.3604, -0.3725, -0.5138],
        [ 0.9580, -0.7806, -0.9501,  0.4071]], requires_grad=True)
tensor([[-0.7947],
        [ 1.2969],
        [ 0.1652],
        [-0.7777]], requires_grad=True)
tensor([[-0.2894],
        [-2.2473]], grad_fn=<MmBackward>)
tensor(-2.5367, grad_fn=<SumBackward0>)
```

```
d.backward()

print(a.grad)
print(b.grad)
```

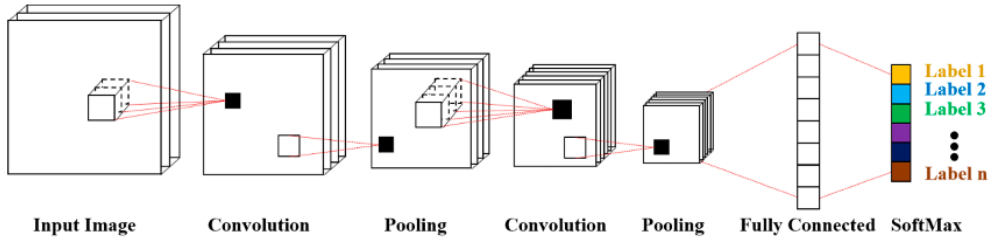
```
tensor([[-0.7947,  1.2969,  0.1652, -0.7777],
        [-0.7947,  1.2969,  0.1652, -0.7777]])
tensor([[ 2.3356],
        [-0.4202],
        [-1.3226],
        [-0.1067]])
```

```
if torch.cuda.is_available():
    model = model.to('cuda')
```

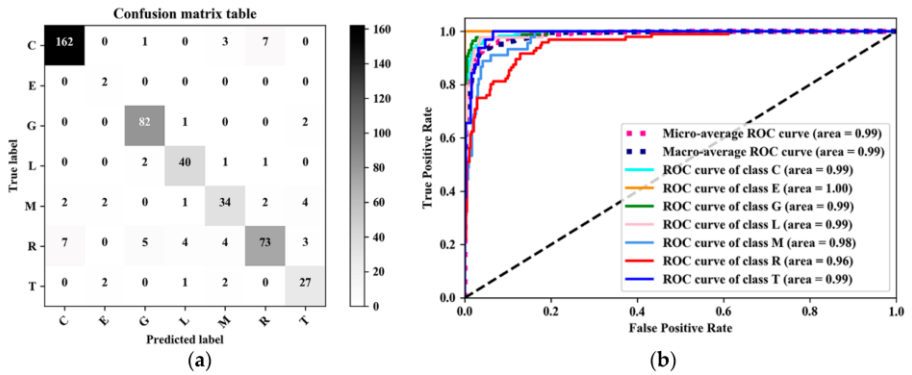
PyTorch

# Что нужно сделать?

Создать и обучить классификатор рентгеновских снимков для диагностирования пневмонии.




Произвести оценку модели




Представить свое решение с кодом для обучения итоговой модели :

- Обосновать выбор архитектуры
- Описать особенности процесса обучения и подготовки данных
- Привести данные об оценке полученной модели
- Опубликовать свое решение на Kaggle*.


Pneumonia Detection using CNN(92.6% Accuracy) | Kaggle