**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по практическим заданиям**

**по дисциплине «САПР РЭА»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 1301 |  | Ахметзянов Д. А. |
|  |  | Герасимов А.М. |
| Преподаватель |  | Боброва. Ю. О. |

Санкт-Петербург

2024

1. **Базовые операции и синтаксис языка Python**

**Цель работы.**

Изучить основы работы с языком python, научиться обрабатывать данные из файла.

**Ход выполнения работы.**

1. Загрузим файл с данными в виде exel таблицы
2. Прочитаем файл с помощью python и выведем его содержимое

import csv

with open('diabetes\_data\_upload.csv', *newline*='') as csvfile:

 data = list(csv.reader(csvfile))

for row in data:

  print(row)

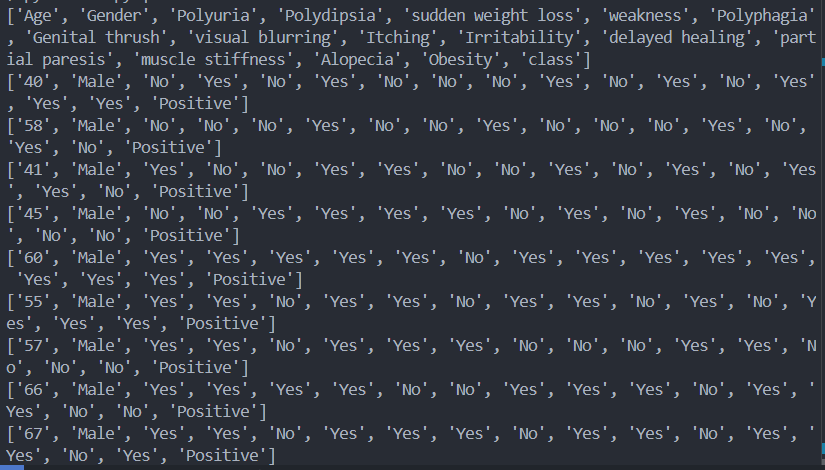


Рис. 1 Вывод таблицы

1. Расчет среднего значения по колонке Age с помощью цикла for

header = data[0]

print("Header:", header)

age\_column\_index = None

for i, column\_name in enumerate(header):

    if 'age' in column\_name.lower():

        age\_column\_index = i

        break

ages = []

for i in range(1, len(data)):

    ages.append(float(data[i][age\_column\_index]))

sum\_num = 0

for t in ages:

    sum\_num = sum\_num + t

avg = sum\_num / len(ages)

print(f"Average age: {avg}")



Рис. 2 Вывод среднего возраста

1. Расчет среднего значения по колонке Age отдельно для мужчин и женщин с помощью условий if-else

gender\_column\_index = None

for i, column\_name in enumerate(header):

    if 'gender' in column\_name.lower():

        gender\_column\_index = i

        break

male\_ages = []

female\_ages = []

for i in range(1, len(data)):

    age = float(data[i][age\_column\_index])

    gender = data[i][gender\_column\_index]

    if gender == "Male":

        male\_ages.append(age)

    elif gender == "Female":

        female\_ages.append(age)

male\_sum = sum(male\_ages)

male\_avg = male\_sum / len(male\_ages)

female\_sum = sum(female\_ages)

female\_avg = female\_sum / len(female\_ages)

print(f"Number of males: {len(male\_ages)}")

print(f"Average age of males: {male\_avg:.2f}")

print(f"Number of females: {len(female\_ages)}")

print(f"Average age of females: {female\_avg:.2f}")

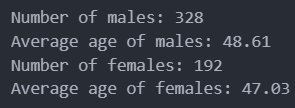


Рис. 3 Вывод информации о количестве представителей каждого пола и их среднем возрасте

1. Построение таблицы, показывающей связь диабета и ожирения в виде связи между классами

class\_column\_index = None

for i, column\_name in enumerate(header):

    if 'class' in column\_name.lower():

        class\_column\_index = i

        break

male\_positive = 0

male\_negative = 0

female\_positive = 0

female\_negative = 0

male\_positive\_indices = []

male\_negative\_indices = []

female\_positive\_indices = []

female\_negative\_indices = []

for i in range(1, len(data)):

    gender = data[i][gender\_column\_index]

    diabetes\_class = data[i][class\_column\_index]

    if gender == "Male" and diabetes\_class == "Positive":

        male\_positive += 1

        male\_positive\_indices.append(i)

    elif gender == "Male" and diabetes\_class == "Negative":

        male\_negative += 1

        male\_negative\_indices.append(i)

    elif gender == "Female" and diabetes\_class == "Positive":

        female\_positive += 1

        female\_positive\_indices.append(i)

    elif gender == "Female" and diabetes\_class == "Negative":

        female\_negative += 1

        female\_negative\_indices.append(i)

total\_patients = male\_positive + male\_negative + female\_positive + female\_negative

male\_positive\_percent = (male\_positive / total\_patients) \* 100

male\_negative\_percent = (male\_negative / total\_patients) \* 100

female\_positive\_percent = (female\_positive / total\_patients) \* 100

female\_negative\_percent = (female\_negative / total\_patients) \* 100

print(f"Total number of patients: {total\_patients}")

print("\nPatient distribution:")

print(f"Male patients with diabetes: {male\_positive} ({male\_positive\_percent:.2f}%)")

print(f"Male patients without diabetes: {male\_negative} ({male\_negative\_percent:.2f}%)")

print(f"Female patients with diabetes: {female\_positive} ({female\_positive\_percent:.2f}%)")

print(f"Female patients without diabetes: {female\_negative} ({female\_negative\_percent:.2f}%)")

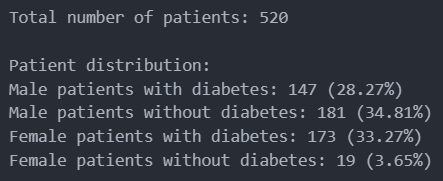


Рис. 4 Вывод данных о количестве пациентов страдающих диабетом по полам

**2. ЗАГРУЗКА И НАСТРОЙКА МОДУЛЕЙ NUMPY, PANDAS, SCIKIT-LEARN, TENSORFLOW, KERAS**

1. Загрузим все необходимые библиотеки

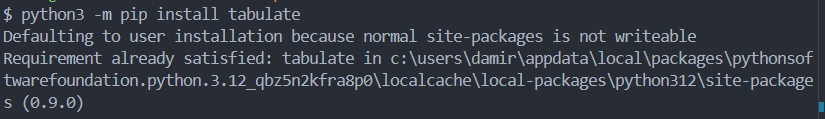


Рис. 5 Вывод консоли после установки зависимости

# Пошаговое написание кода

import numpy as np

import pandas as pd

import sklearn

import tensorflow as tf

import keras

print(f"NumPy version: {np.\_\_version\_\_}")

print(f"Pandas version: {pd.\_\_version\_\_}")

print(f"Scikit-learn version: {sklearn.\_\_version\_\_}")

print(f"TensorFlow version: {tf.\_\_version\_\_}")

print(f"Keras version: {keras.\_\_version\_\_}")

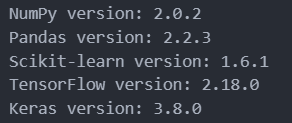


Рис. 6 Версии установленных пакетов

table = pd.read\_csv('diabetes\_data\_upload.csv')

print(table.head())

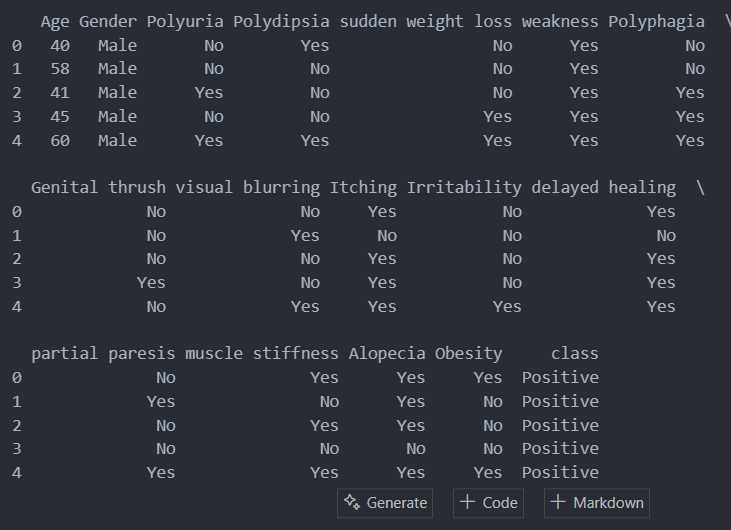


Рис. 7 Вывод загруженных данных, прочитанных с помощью pandas

mean\_age\_all = table['Age'].mean()

mean\_age\_by\_gender = table.groupby('Gender')['Age'].mean()

print("Using pandas:")

print(f"Mean age (all): {mean\_age\_all:.2f}")

print(f"Mean age by gender:\n{mean\_age\_by\_gender}")

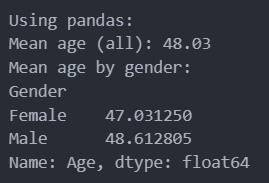


Рис. 8 Данные о пациентах полученные с помощью pandas

Данные совпадают с рассчитанными ранее с помощью нативных средств python.

С помощью модуля time измерим время затраченное на рассчеты с помощью pandas и без его использования

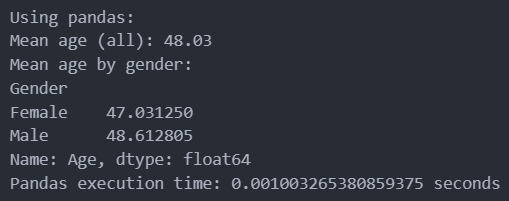


Рис. 9 Время затраченное на рассчеты с помощью pandas

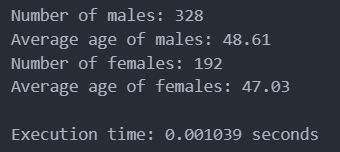


Рис. 10 Время затраченное на рассчеты без pandas

Как видно pandas оказался незначительно быстрее

**3. NUMPY ARRAYS AND FUNCTIONS**

1. Создадим numpy массив где длина равна моему номеру в списке группы (2), а ширина номеру первой буквы моей фамилии (1).

import numpy as np

array = np.zeros((2, 1))

print("Массив:")

print(array)

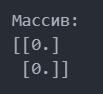


Рис. 11 Итоговый массив

1. Создадим массивы нужного размера, заполненные случайными числами распределенными равномерно и нормально

uniform\_array = np.random.rand(2, 1)

print("Массив с равномерным распределением:")

print(uniform\_array)

normal\_array = np.random.randn(2, 1)

print("Массив с нормальным распределением:")

print(normal\_array)

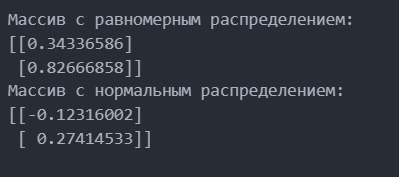


Рис. 12 Созданные массивы

1. Изучим характеристики созданных массивов с помощью стандартных средств numpy

print(f"ndim: {array.ndim}")

print(f"shape: {array.shape}")

print(f"size: {array.size}")

print(f"dtype: {array.dtype}")

print(f"itemsize: {array.itemsize} bytes")

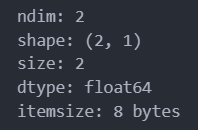


Рис. 13 Характеристики массива

1. Создадим функцию выводящую свойства массива в консоль

def print\_array\_info(*a*):

    print(f"Shape: {*a*.shape}")

    print(f"Dimensions: {*a*.ndim}")

    print(f"Data type: {*a*.dtype.name}")

    print(f"Item size: {*a*.itemsize} bytes")

    print(f"Total elements: {*a*.size}")

1. Откроем загруженный файл с помощью pandas и определим типы его столбцов

import pandas as pd

df = pd.read\_csv('diabetes\_data\_upload.csv')

dtypes = {}

for column in df.columns:

    numpy\_type = df[column].to\_numpy().dtype

    dtypes[column] = str(numpy\_type)

dtypes\_df = pd.DataFrame(list(dtypes.items()), *columns*=['Column', 'Data\_Type'])

dtypes\_df.to\_csv('diabetes\_data\_types.csv', *index*=False)

print(dtypes\_df)

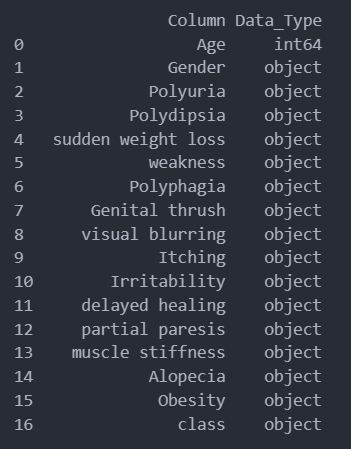


Рис. 14 Типы столбцов таблицы

1. Переконвертируем входную таблицу так чтобы таблица не содержала буквенных обозначений классов.

df\_numeric = df.copy()

df\_numeric['Gender'] = np.where(df\_numeric['Gender'] == 'Male', 1, 0)

yes\_no\_columns = ['Polyuria', 'Polydipsia', 'sudden weight loss', 'weakness',

                  'Polyphagia', 'Genital thrush', 'visual blurring', 'Itching',

                  'Irritability', 'delayed healing', 'partial paresis',

                  'muscle stiffness', 'Alopecia', 'Obesity']

for col in yes\_no\_columns:

    df\_numeric[col] = np.where(df\_numeric[col] == 'Yes', 1, 0)

df\_numeric['class'] = np.where(df\_numeric['class'] == 'Positive', 1, 0)

print(df\_numeric.head())

df\_numeric.to\_csv('diabetes\_data\_numeric.csv', *index*=False)

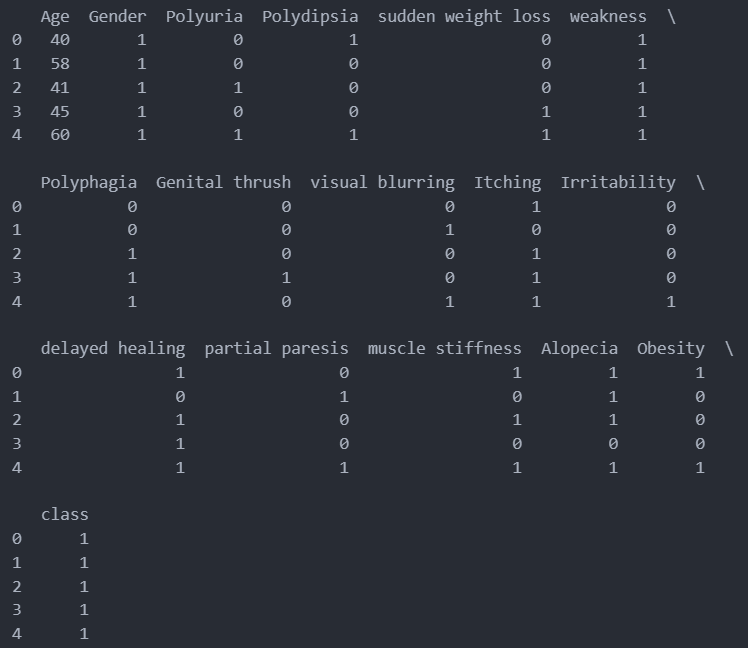


Рис. 15 Новая таблица

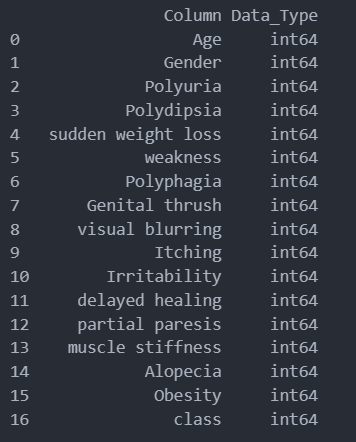


Рис. 16 Типы столбцов после преобразования

**4. РАБОТА С ТАБЛИЧНЫМИ ДАННЫМИ В БИБЛИОТЕКЕ PANDAS**

1. Напишем функцию которая выводит в консоль характеристики датафрейма

def analyze\_dataframe(*df*):

    n\_cols = 2 + 2

    selected\_df = *df*.iloc[:, :n\_cols]

    print(f"Index: {type(*df*.index)}")

    print()

    print("Types of columns:")

    print(selected\_df.dtypes)

    print()

    print("Summary of columns:")

    print(selected\_df.describe())

    print()

    print("First 5 rows:")

    print(selected\_df.head())

Вызовем эту функцию к нашей таблице

import pandas as pd

df = pd.read\_csv('diabetes\_data\_upload.csv')

analyze\_dataframe(df)

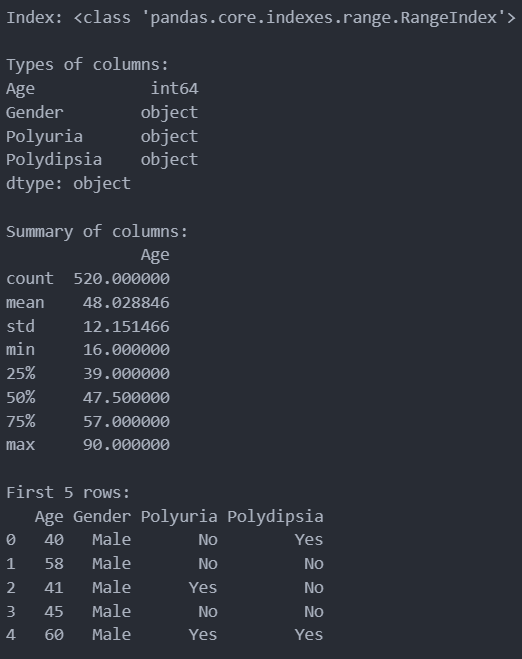


Рис. 17 Статистика по таблице

1. Создадим два датафрейма, в зависимости от содержимого столбца N+1

df\_yes = df[df['Polyuria'] == 'Yes']

df\_no = df[df['Polyuria'] == 'No']

print(f"Yes: {df\_yes.shape[0]} rows")

print(f"No: {df\_no.shape[0]} rows")

print("\nFirst 5 rows of df\_yes:")

print(df\_yes.head())

print("\nFirst 5 rows of df\_no:")

print(df\_no.head())

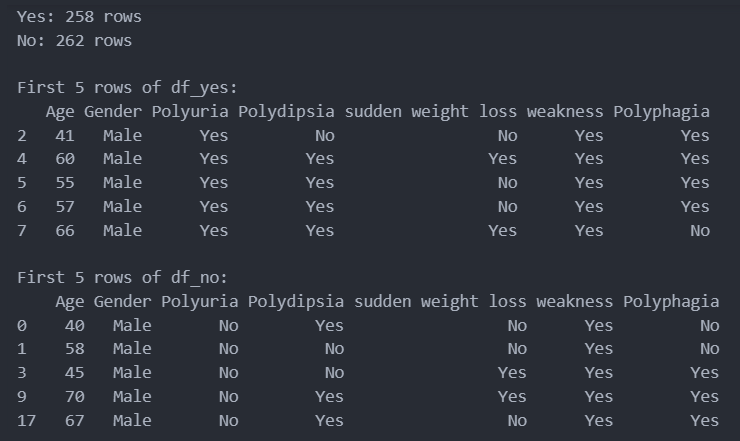


Рис. 18 Вывод программы

1. Отсортируем таблицу по нескольким столбцам

sorted\_df = df.sort\_values(*by*=['Polyuria', 'Polydipsia', 'Age'])

print(sorted\_df[['Polyuria', 'Polydipsia', 'Age']].head())

print(sorted\_df[['Polyuria', 'Polydipsia', 'Age']].tail())

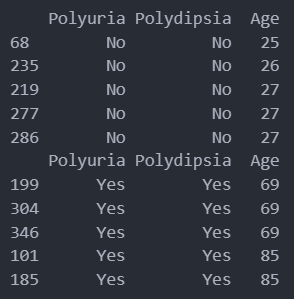


Рис. 19 head и tail отсортированной таблицы

1. Проверим наши данные на наличие пропусков

print("Missing values per column:")

print(df.isnull().sum())

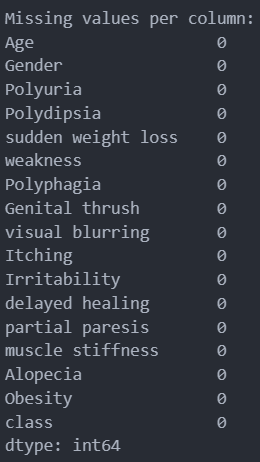


Рис. 20 Наличие пропусков по каждой колонке

1. Построим графики по первым трем колонкам

import pandas as pd

import matplotlib.pylab as plt

df = pd.read\_csv('diabetes\_data\_upload.csv')

df1 = df.iloc[:,0:3]

plt.hist(df1.iloc[:,0])

plt.show()

plt.hist(df1.iloc[:,1])

plt.show()

plt.hist(df1.iloc[:,2])

plt.show()

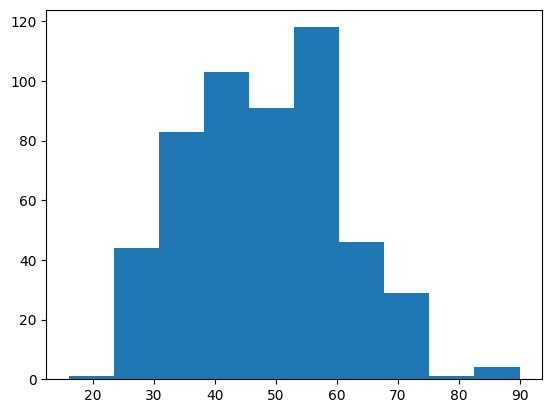


Рис. 21 Диаграмма распределения возраста

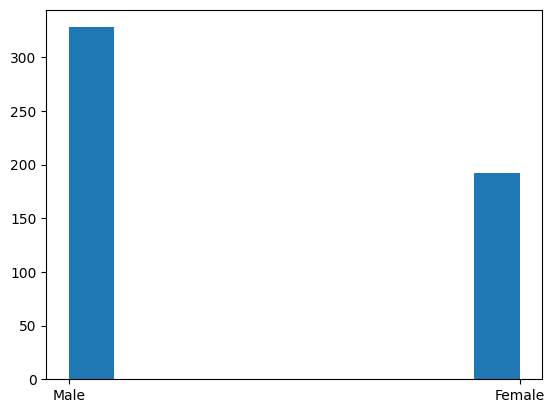


Рис. 22 Диаграмма распределения полов

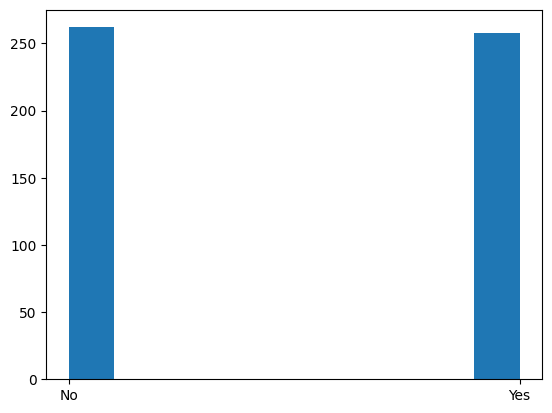


Рис. 23 Диаграмма распределения параметра Polyuria

1. Построим две диаграммы распределения age

import numpy as np

import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, *figsize*=(12, 5), *sharey*=True)

axes[0].hist(df\_yes['Age'], *bins*=10, *color*='blue', *alpha*=0.7, *label*='Polyuria: Yes')

axes[1].hist(df\_no['Age'], *bins*=10, *color*='green', *alpha*=0.7, *label*='Polyuria: No')

axes[0].set\_title('Age Distribution - Polyuria: Yes')

axes[0].set\_xlabel('Age')

axes[0].set\_ylabel('Frequency')

axes[1].set\_title('Age Distribution - Polyuria: No')

axes[1].set\_xlabel('Age')

axes[0].legend()

axes[1].legend()

plt.tight\_layout()

plt.show()

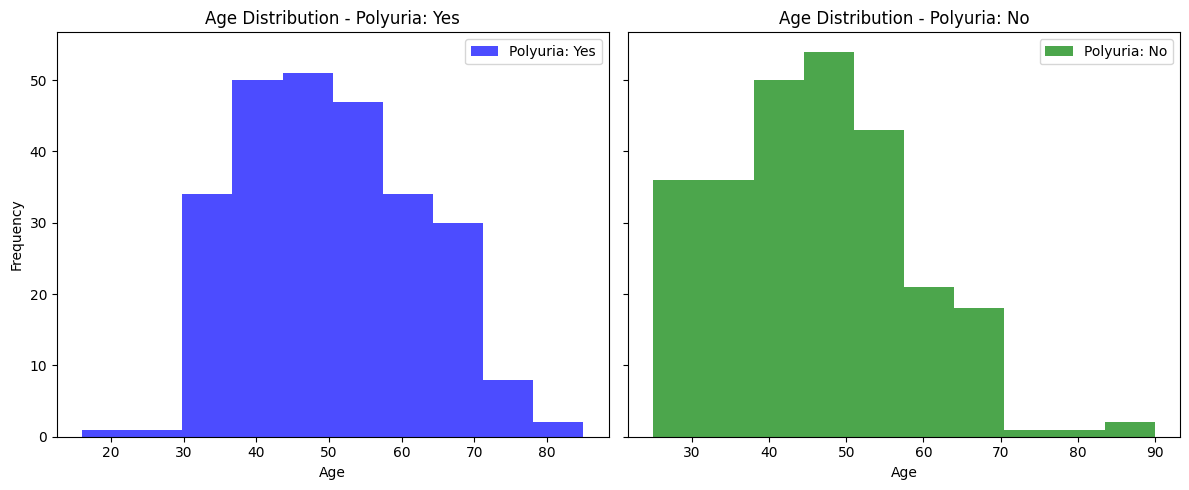


Рис. 24 Распределение age в зависимости от наличия полиурии

Как видно полиурии незначительно подвержены люди младше 30 лет, в основном люди с полиурией находятся в диапазоне 30-70 лет.

1. Построим boxplot распределения age этих двух таблиц

import matplotlib.pyplot as plt

plt.figure(*figsize*=(10, 6))

plt.boxplot([df\_yes['Age'], df\_no['Age']], *labels*=['Polyuria: Yes', 'Polyuria: No'], *widths*=0.5)

plt.title('Age Distribution Comparison by Polyuria', *fontsize*=14)

plt.ylabel('Age', *fontsize*=12)

plt.grid(*axis*='y', *linestyle*='--', *alpha*=0.7)

plt.show()

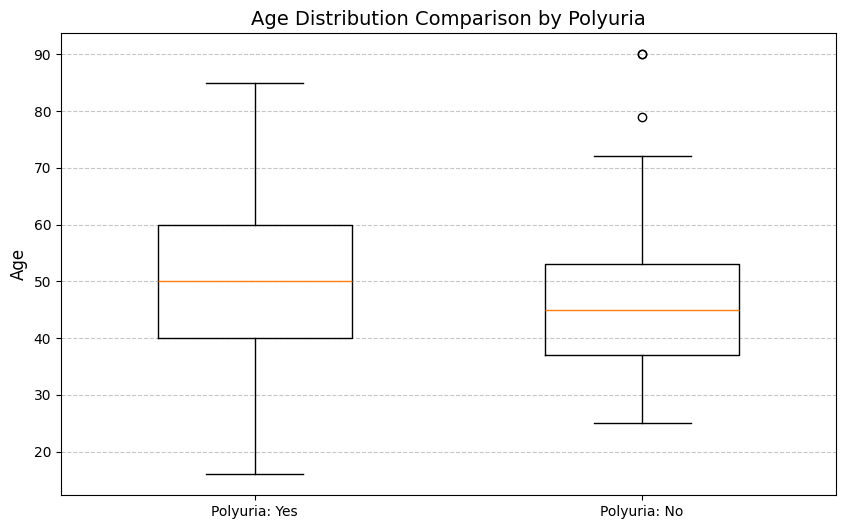


Рис. 25 Boxplot распределения age

Как видно, люди больные полиурией в среднем немного старше, а также имеют больший возрастной разброс чем люди без нее.

1. Построим scatter matrix для колонок 3 и 4

*# No need to import pandas again or read the CSV again as these are already done*

import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 1)

pd.plotting.scatter\_matrix(df[['Age', 'Polyuria', 'Polydipsia']],

*c*=['red' if c == 1 else 'blue' for c in df['class']], *alpha*=0.1, *ax*=axes, *diagonal*='hist')

plt.tight\_layout()

plt.show()

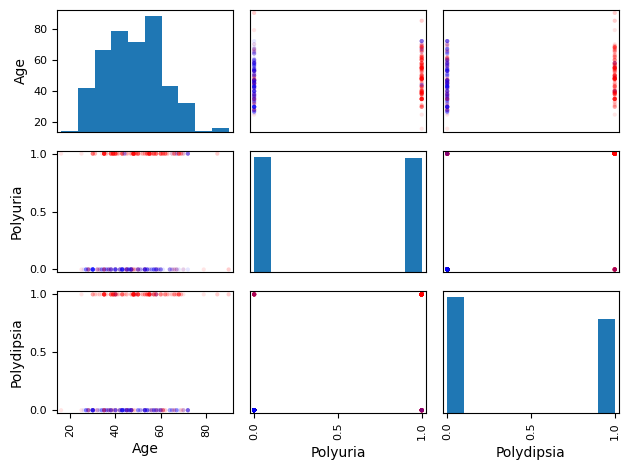


Рис. 26 scatter matrix

1. **МАШИННОЕ ОБУЧЕНИЕ С БИБЛИОТЕКОЙ SCIKITLEARN**
2. Разделим таблицу на X и Y

X = df.drop(*columns*=['class'])

Y = df['class']

1. Стандартизируем значения колонки Age

X['Age'] = sklearn.preprocessing.scale(X['Age'])

1. Разделим выборку на тренировочную и тестовую

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, *test\_size*=0.2, *random\_state*=2)

print(X\_train.shape, Y\_train.shape)

print(X\_test.shape, Y\_test.shape)

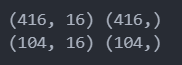


Рис. 27 Размеры выборок

1. Обучим модель с помощью логистической регрессии

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(*random\_state*=0).fit(X\_train, Y\_train)

1. Используя метод predict предскажем значения для тестовой выборки

Y\_pred = model.predict(X\_test)

1. Оценим различные метрики точности модели

print(accuracy\_score(Y\_test, Y\_pred))

Полученное значение 91.3% точности

Можно считать что мы получили довольно высокую степень точности, однако в сфере медицины в идеале должно быть как можно меньше ложно негативных результатов, в данном случае их 10%.

1. Попробуем улучшить результаты модели с помощью параметра C

import numpy as np

max\_acc = 0

max\_acc\_c = 100

param\_range = np.logspace(2, -3, 6)

for param in param\_range:

    Y\_pred = LogisticRegression(*random\_state*=0, *C*=param).fit(X\_train, Y\_train).predict(X\_test)

    acc = accuracy\_score(Y\_test, Y\_pred)

    print(acc, param)

    if acc > max\_acc:

        max\_acc = acc

        max\_acc\_c = param

print(max\_acc)

print(max\_acc\_c)

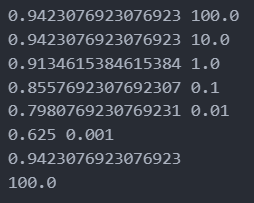


Рис. 28 Результаты в зависимости от параметра

Как можно заметить лучшая точность была получена для значения параметра больше 10 - 94.2%

**6. СОЗДАНИЕ КЛАССИФИКАТОРА С ПОМОЩЬЮ БИБЛИОТЕКИ PYTORCH**

1. Сгенерируем тестовый датасет и отобразим его

from sklearn.datasets import make\_moons

from matplotlib import pyplot as plt

X, y = make\_moons(*n\_samples*=5000, *random\_state*=1, *noise*=0.1)

plt.figure(*figsize*=(16, 10))

plt.title("Dataset")

plt.scatter(X[:, 0], X[:, 1], *c*=y, *cmap*="summer")

plt.show()

1. Разделим данные на тестовую и тренировочную выборки

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=2)

print(X\_train.shape, Y\_train.shape)

print(X\_test.shape, Y\_test.shape)

1. Загрузим датасет в pytorch

import torch

from torch.utils.data import TensorDataset, DataLoader

X\_train\_t = torch.FloatTensor(X\_train)

y\_train\_t = torch.FloatTensor(Y\_train)

X\_val\_t = torch.FloatTensor(X\_test)

y\_val\_t = torch.FloatTensor(Y\_test)

train\_dataset = TensorDataset(X\_train\_t, y\_train\_t)

val\_dataset = TensorDataset(X\_val\_t, y\_val\_t)

train\_dataloader = DataLoader(train\_dataset, *batch\_size*=128)

val\_dataloader = DataLoader(val\_dataset, *batch\_size*=128)

1. Создадим простейший класификатор

from torch import nn

class LinearRegression(nn.Module):

 def \_\_init\_\_(*self*, *in\_features*: int, *out\_features*: int, *bias*: bool = True):

    super().\_\_init\_\_()

*self*.weights = nn.Parameter(torch.Tensor(*in\_features*, *out\_features*))

*self*.bias = *bias*

    if *bias*:

*self*.bias\_term = nn.Parameter(torch.randn(*out\_features*))

 def forward(*self*, *x*):

*x* = *x*@*self*.weights

    if *self*.bias:

*x* += *self*.bias\_term

    return *x*

1. Зададим функцию потерь и метод оптимизации

linear\_regression = LinearRegression(2, 1)

loss\_function = nn.BCEWithLogitsLoss()

optimizer = torch.optim.SGD(linear\_regression.parameters(), *lr*=0.01\*2)

list(linear\_regression.parameters())

1. Создадим основной цикл обучения модели

N = 2

tol = 1e-5

losses = []

max\_epochs = 5\*N

prev\_weights = torch.zeros\_like(linear\_regression.weights)

stop\_it = False

for epoch in range(max\_epochs):

    for it, (X\_batch, y\_batch) in enumerate(train\_dataloader):

        optimizer.zero\_grad()

        outp = linear\_regression.forward(X\_batch).squeeze(1)

        loss = loss\_function(outp, y\_batch)

        loss.backward()

        losses.append(loss.detach().flatten()[0])

        optimizer.step()

        probabilities = linear\_regression.forward(X\_batch)

        preds = (probabilities > 0.5).type(torch.long)

        batch\_acc = (preds.flatten() == y\_batch).type(torch.float32).sum()/y\_batch.size(0)

        if it % 10 == 0:

            print(f"Iteration: {it + epoch\*len(train\_dataloader)}\nBatch accuracy: {batch\_acc}")

        current\_weights = linear\_regression.weights.detach().clone()

        if (prev\_weights - current\_weights).abs().max() < tol:

            print(f"\nIteration: {it + epoch\*len(train\_dataloader)}. Convergence. Stopping iterations.")

            stop\_it = True

            break

        prev\_weights = current\_weights

    if stop\_it:

        break

1. Построим график зависимости знчения потерь от итерации

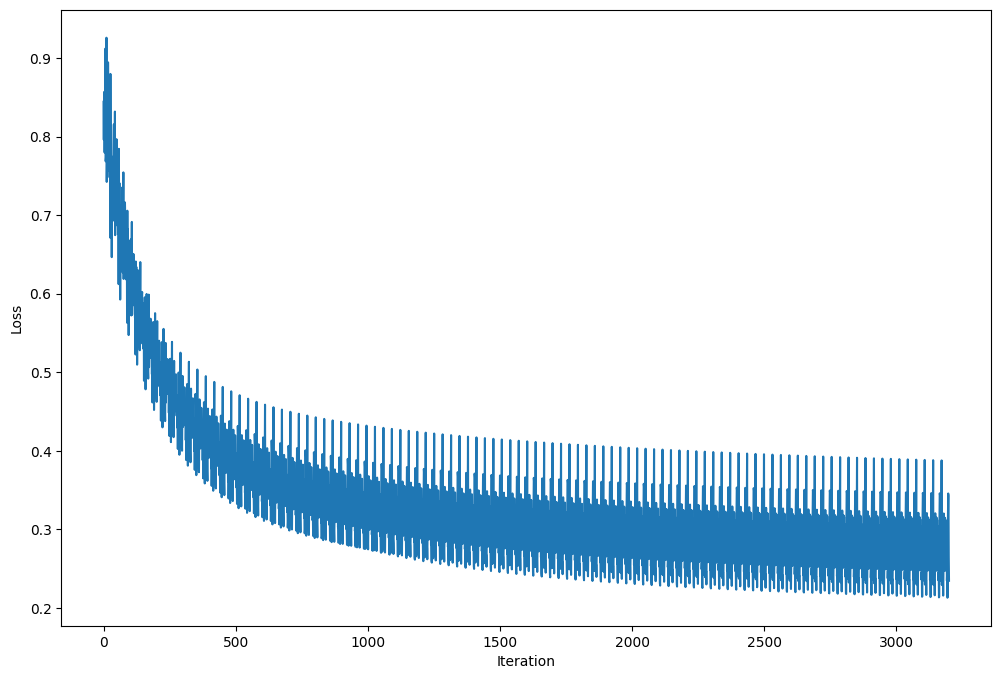


Рис. 29 График зависимости потерь от итерации

1. Измерим точность полученной модели

import numpy as np

def predict(*dataloader*, *model*):

*model*.eval()

    predictions = np.array([])

    for x\_batch, \_ in *dataloader*:

        outp = *model*(x\_batch)

        probs = torch.sigmoid(outp)

        preds = (probs > 0.5).type(torch.long)

        predictions = np.hstack((predictions, preds.numpy().flatten()))

    return predictions.flatten()

from sklearn.metrics import accuracy\_score

print(accuracy\_score(predict(val\_dataloader, linear\_regression),Y\_test))

Результат 87.8%

1. Значительно улучшить точность модели путем подбора параметров и функции потерь не получилось

**7. Сверточные нейронные сети pytorch для классификации изображений**

1. Создадим простейшую сверточную нейронную сеть

N = 2

import os

import torch

import torchvision

from torch.utils.data import TensorDataset, DataLoader

from catalyst import utils

from catalyst.contrib.datasets import MNIST

from torch import nn

import six

if not hasattr(six, 'string\_classes'):

    if hasattr(six, 'string\_types'):

        six.string\_classes = six.string\_types

    else:

*# For newer Python versions*

        six.string\_classes = (str,)

*# Add this to ensure torch.\_six has string\_classes*

import torch

if not hasattr(torch, '\_six'):

    torch.\_six = six

elif not hasattr(torch.\_six, 'string\_classes'):

    torch.\_six.string\_classes = six.string\_classes

if not hasattr(torch, '\_six'):

    torch.\_six = six

utils.set\_global\_seed(N)

train\_dataset = MNIST(*root*=os.getcwd(), *train*=True, *download*=True)

val\_dataset = MNIST(*root*=os.getcwd(), *train*=False)

train\_dataloader = DataLoader(train\_dataset, *batch\_size*=128)

val\_dataloader = DataLoader(val\_dataset, *batch\_size*=128)

class Identical(nn.Module):

    def forward(*self*, *x*):

        return *x*

class Flatten(nn.Module):

    def forward(*self*, *x*):

        batch\_size = *x*.size(0)

        return *x*.view(batch\_size, -1)

activation = Identical

model = nn.Sequential(

 Flatten(),

 nn.Linear(28\*28, 128),

 activation(),

 nn.Linear(128, 128),

 activation(),

 nn.Linear(128, 10)

)

1. Зададим настройки обучения

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters())

loaders = {"train": train\_dataloader, "valid": val\_dataloader}

1. Напишем основной цикл обучения нейросети

max\_epochs = N \* 5

accuracy = {"train": [], "valid": []}

for epoch in range(max\_epochs):

    epoch\_correct = 0

    epoch\_all = 0

    for k, dataloader in loaders.items():

        epoch\_correct = 0

        epoch\_all = 0

        for x\_batch, y\_batch in dataloader:

            if k == "train":

                model.train()

                optimizer.zero\_grad()

                outp = model(x\_batch.float().unsqueeze(1))

            else:

                model.eval()

                with torch.no\_grad():

                    outp = model(x\_batch.float().unsqueeze(1))

            preds = outp.argmax(-1)

            correct = (preds == y\_batch).sum()

            all = len(y\_batch)

            epoch\_correct += correct.item()

            epoch\_all += all

            if k == "train":

                loss = criterion(outp, y\_batch)

                loss.backward()

                optimizer.step()

        print(f"Epoch: {epoch+1}")

        print(f"Loader: {k}. Accuracy: {epoch\_correct/epoch\_all}")

        accuracy[k].append(epoch\_correct/epoch\_all)

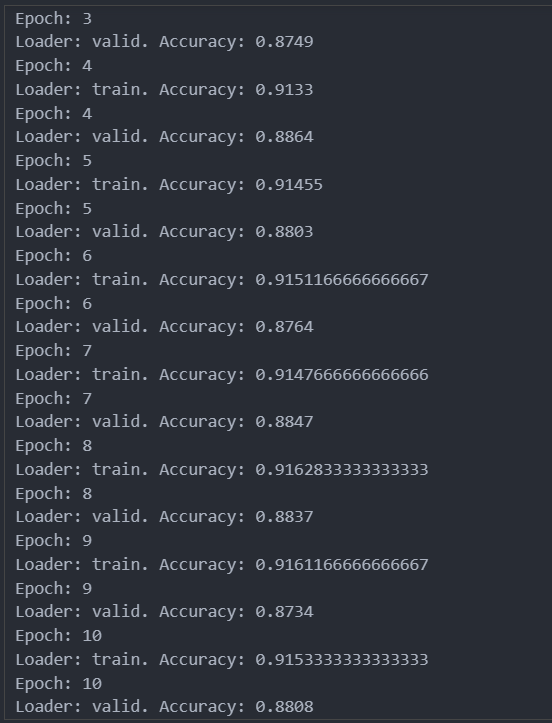


Рис. 30 Результаты тренировки модели

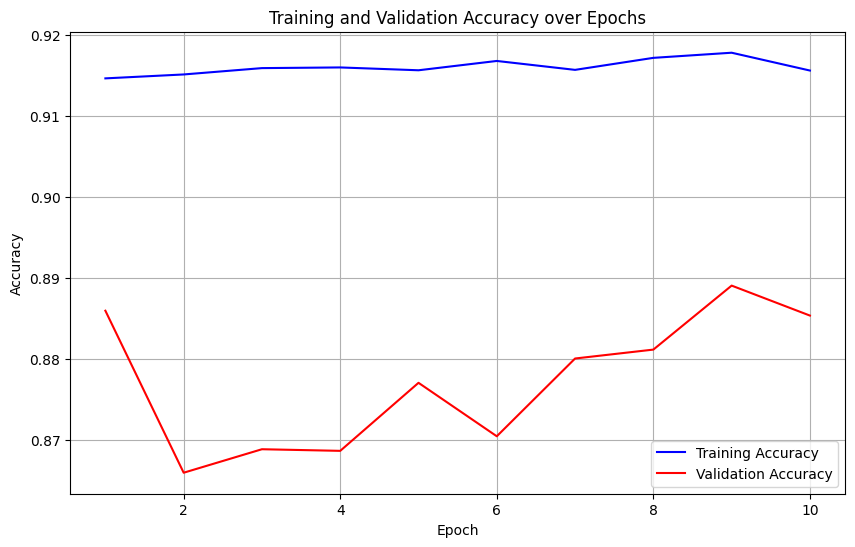


Рис. 31 Зависимость точности модели от эпохи

Как видно в конце обучения модель обладает точностью 89%

1. Улучшим качество прогнозирования за счет изменения параметров модели

N = 2

import os

import torch

import torchvision

from torch.utils.data import TensorDataset, DataLoader

from catalyst import utils

from catalyst.contrib.datasets import MNIST

from torch import nn

utils.set\_global\_seed(N)

import six

if not hasattr(six, 'string\_classes'):

    if hasattr(six, 'string\_types'):

        six.string\_classes = six.string\_types

    else:

*# For newer Python versions*

        six.string\_classes = (str,)

*# Add this to ensure torch.\_six has string\_classes*

import torch

if not hasattr(torch, '\_six'):

    torch.\_six = six

elif not hasattr(torch.\_six, 'string\_classes'):

    torch.\_six.string\_classes = six.string\_classes

if not hasattr(torch, '\_six'):

    torch.\_six = six

import torch.nn.functional as F

utils.set\_global\_seed(N)

train\_dataset = MNIST(*root*=os.getcwd(), *train*=True, *download*=True)

val\_dataset = MNIST(*root*=os.getcwd(), *train*=False)

train\_dataloader = DataLoader(train\_dataset, *batch\_size*=128)

val\_dataloader = DataLoader(val\_dataset, *batch\_size*=128)

class Identical(nn.Module):

    def forward(*self*, *x*):

        return *x*

class Flatten(nn.Module):

    def forward(*self*, *x*):

        batch\_size = *x*.size(0)

        return *x*.view(batch\_size, -1)

activation = Identical

class Flatten(nn.Module):

    def forward(*self*, *x*):

        batch\_size = *x*.size(0)

        return *x*.view(batch\_size, -1)

class EnhancedMNISTModel(nn.Module):

    def \_\_init\_\_(*self*, *dropout\_rate*=0.2):

        super(EnhancedMNISTModel, *self*).\_\_init\_\_()

*self*.flatten = Flatten()

*self*.fc1 = nn.Linear(28\*28, 512)

*self*.bn1 = nn.BatchNorm1d(512)

*self*.dropout1 = nn.Dropout(*dropout\_rate*)

*self*.fc2 = nn.Linear(512, 256)

*self*.bn2 = nn.BatchNorm1d(256)

*self*.dropout2 = nn.Dropout(*dropout\_rate*)

*self*.fc3 = nn.Linear(256, 128)

*self*.bn3 = nn.BatchNorm1d(128)

*self*.dropout3 = nn.Dropout(*dropout\_rate*)

*self*.fc4 = nn.Linear(128, 10)

    def forward(*self*, *x*):

*x* = *self*.flatten(*x*)

*x* = F.relu(*self*.bn1(*self*.fc1(*x*)))

*x* = *self*.dropout1(*x*)

*x* = F.relu(*self*.bn2(*self*.fc2(*x*)))

*x* = *self*.dropout2(*x*)

*x* = F.relu(*self*.bn3(*self*.fc3(*x*)))

*x* = *self*.dropout3(*x*)

*x* = *self*.fc4(*x*)

        return *x*

model = EnhancedMNISTModel()

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.AdamW(model.parameters(), *lr*=0.001, *weight\_decay*=1e-5)

scheduler = torch.optim.lr\_scheduler.ReduceLROnPlateau(

    optimizer, 'max', *patience*=3, *factor*=0.5, *verbose*=True

)

loaders = {"train": train\_dataloader, "valid": val\_dataloader}

max\_epochs = N \* 5

accuracy = {"train": [], "valid": []}

for epoch in range(max\_epochs):

    epoch\_correct = 0

    epoch\_all = 0

    for k, dataloader in loaders.items():

        epoch\_correct = 0

        epoch\_all = 0

        for x\_batch, y\_batch in dataloader:

            if k == "train":

                model.train()

                optimizer.zero\_grad()

                x\_batch = x\_batch.float()

                x\_batch = (x\_batch - x\_batch.mean()) / (x\_batch.std() + 1e-8)

                outp = model(x\_batch.float().unsqueeze(1))

            else:

                model.eval()

                with torch.no\_grad():

                    x\_batch = x\_batch.float()

                    x\_batch = (x\_batch - x\_batch.mean()) / (x\_batch.std() + 1e-8)

                    outp = model(x\_batch.unsqueeze(1))

            preds = outp.argmax(-1)

            correct = (preds == y\_batch).sum()

            all = len(y\_batch)

            epoch\_correct += correct.item()

            epoch\_all += all

            if k == "train":

                loss = criterion(outp, y\_batch)

                loss.backward()

                optimizer.step()

        print(f"Epoch: {epoch+1}")

        print(f"Loader: {k}. Accuracy: {epoch\_correct/epoch\_all}")

        accuracy[k].append(epoch\_correct/epoch\_all)

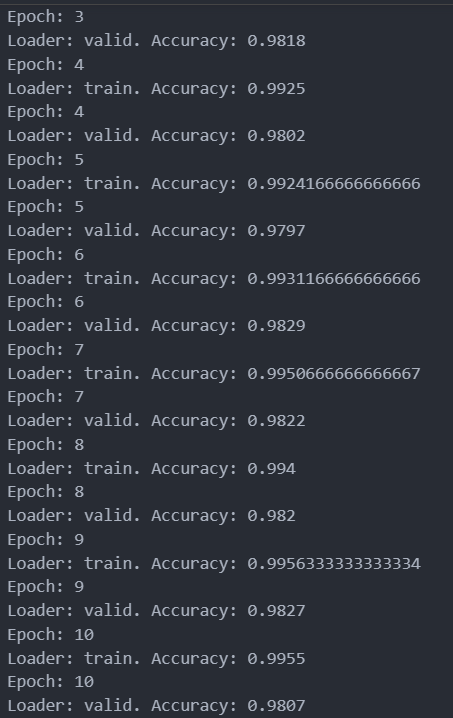


Рис. 32 Результаты обучения модели

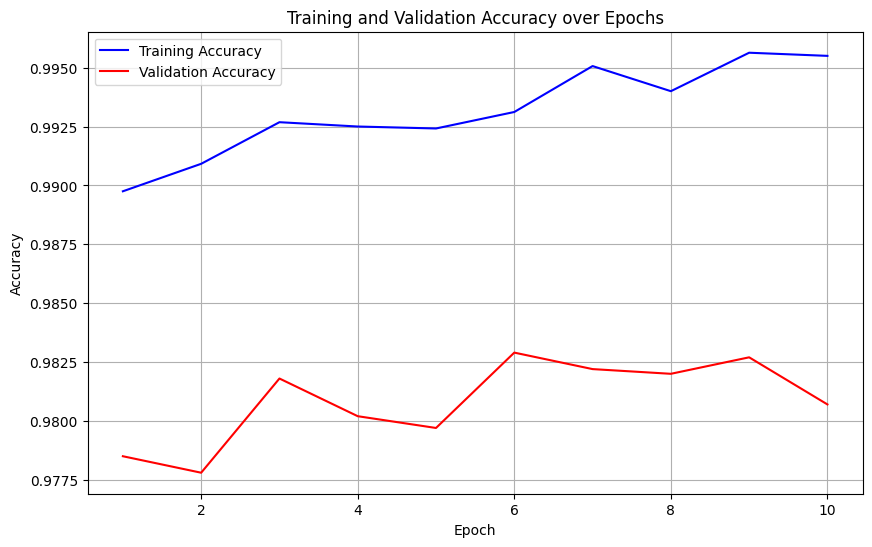


Рис. 33 Точность предсказания модели в зависимости от эпохи

Как видно точность предсказания значительно увеличилась

1. Попробуем использовать предобученную модель для улучшения точности

model = torchvision.models.resnet18(*pretrained*=True)

ct = 0

for child in model.children():

 ct += 1

 if ct < 4:

    for param in child.parameters():

        param.requires\_grad = False

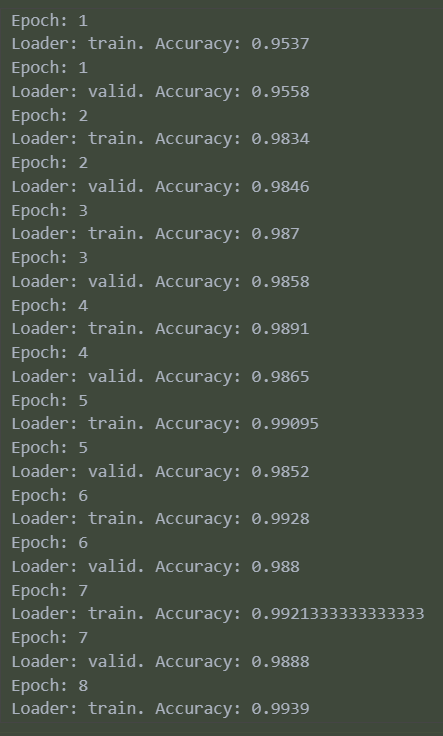


Рис. 34 Результаты преднастроенной модели

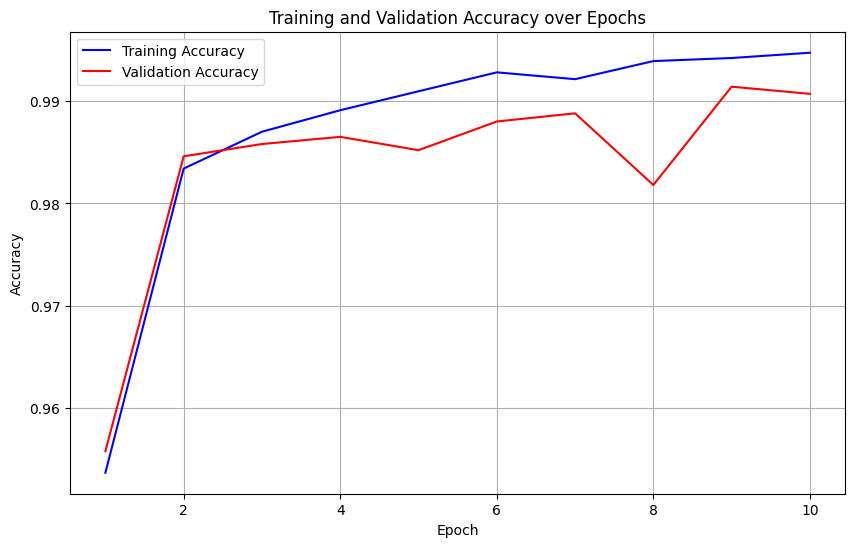


Рис. 35 Точность преднастроенной модели в зависимости от эпохи

Как видно использование преднастроенной модели также положительно отразилось на точности модели, точность очень близка к 100 процентам.

**Вывод**

Мы изучили возможности python и его экосистемы применительно к обработке данных и созданию нейронных сетей. Мы изучили библиотеки numpy и pandas для считывания наборов данных из таблиц, изучили различные способы обработки и аггрегирования данных с помощью этих библиотек.

Также мы получили базовые знания по созданию классификаторов и нейронных сетей с помощью библиотек sklearn, pytorch. Самостоятельно обучили классификатор и нейронную сеть с помощью библиотечных функций. Попробовали улучшить точность моделей с помощью настройки параметров.