МОСКВОСКИЙ
АВИАЦИОННЫЙ
ИНСТИТУТ

# КУРСОВОЙ ПРОЕКТ

## ПО МАШИННОМУ ОБУЧЕНИЮ

МОСКВА, **2023** Г.

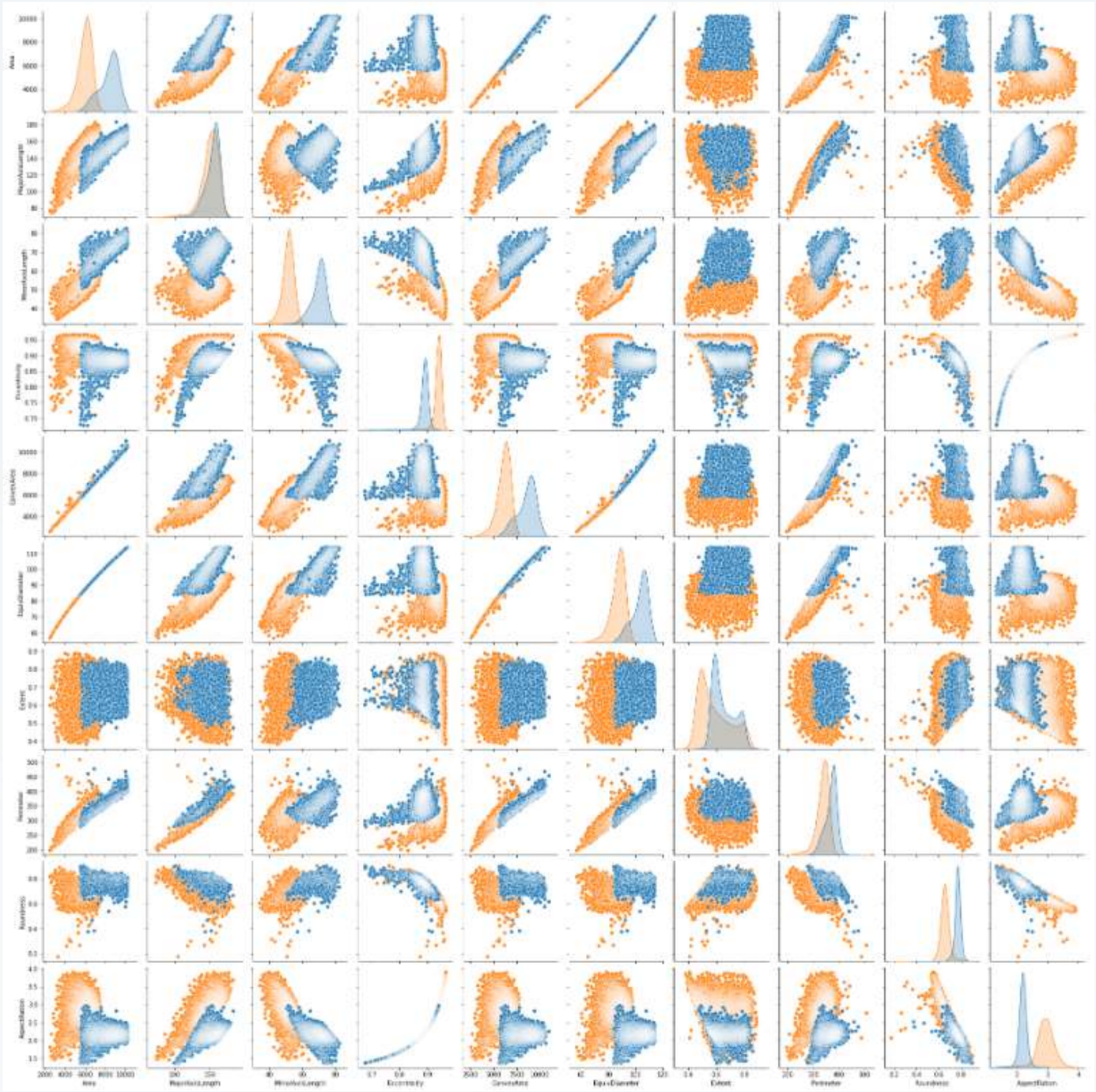
# КОМАНДА ПРОЕКТА

| ФИО | Группа | Роль в команде |
|---|---|---|
| Лагуткина Мария Сергеевна | М8О-109Б-23 | Подготовка модели |
| Савин Александр Андреевич | М8О-114Б-23 | Подготовка данных |
| Спиридонов Кирилл Анатольевич | М8О-107Б-23 | Сбор решения задачи |

# ОПИСАНИЕ ДАТАСЕТА

Датасет **Rice type classification.** Требуется предсказать тип риса **(Jasmine – 1, Gonen – 0)**, основываясь на его признаках (площадь, средняя ширина и длина, скругленность и т.д.).

| | id | Area | MajorAxisLength | MinorAxisLength | Eccentricity | ConvexArea | EquivDiameter | Extent | Perimeter | Roundness | AspectRation | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 4537 | 92.229316 | 64.012769 | 0.719916 | 4677 | 76.004525 | 0.657536 | 273.085 | 0.764510 | 1.440796 | 1 |
| 1 | 2 | 2872 | 74.691881 | 51.400454 | 0.725553 | 3015 | 60.471018 | 0.713009 | 208.317 | 0.831658 | 1.453137 | 1 |
| 2 | 3 | 3048 | 76.293164 | 52.043491 | 0.731211 | 3132 | 62.296341 | 0.759153 | 210.012 | 0.868434 | 1.465950 | 1 |
| 3 | 4 | 3073 | 77.033628 | 51.928487 | 0.738639 | 3157 | 62.551300 | 0.783529 | 210.657 | 0.870203 | 1.483456 | 1 |
| 4 | 5 | 3693 | 85.124785 | 56.374021 | 0.749282 | 3802 | 68.571668 | 0.769375 | 230.332 | 0.874743 | 1.510000 | 1 |

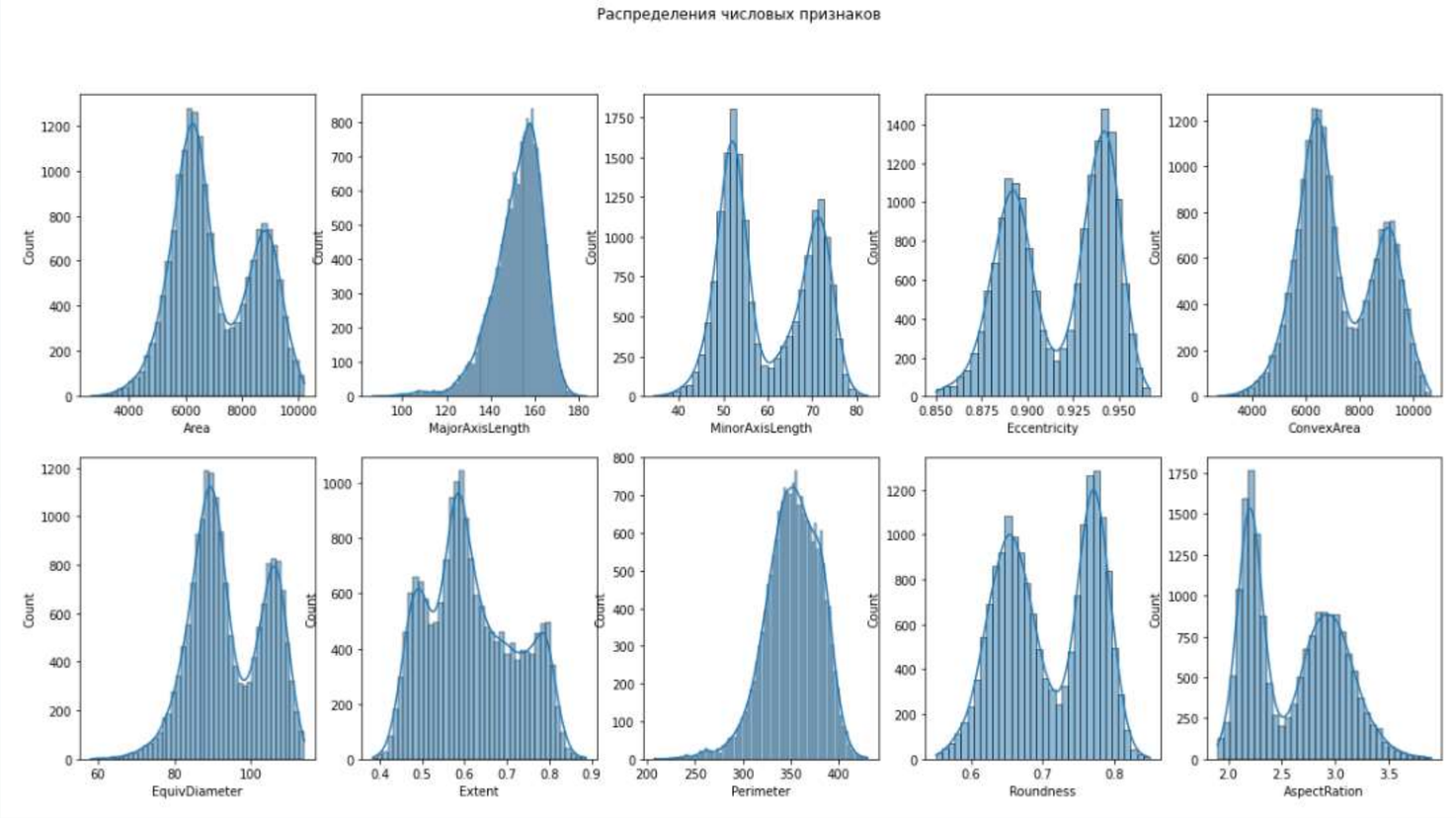


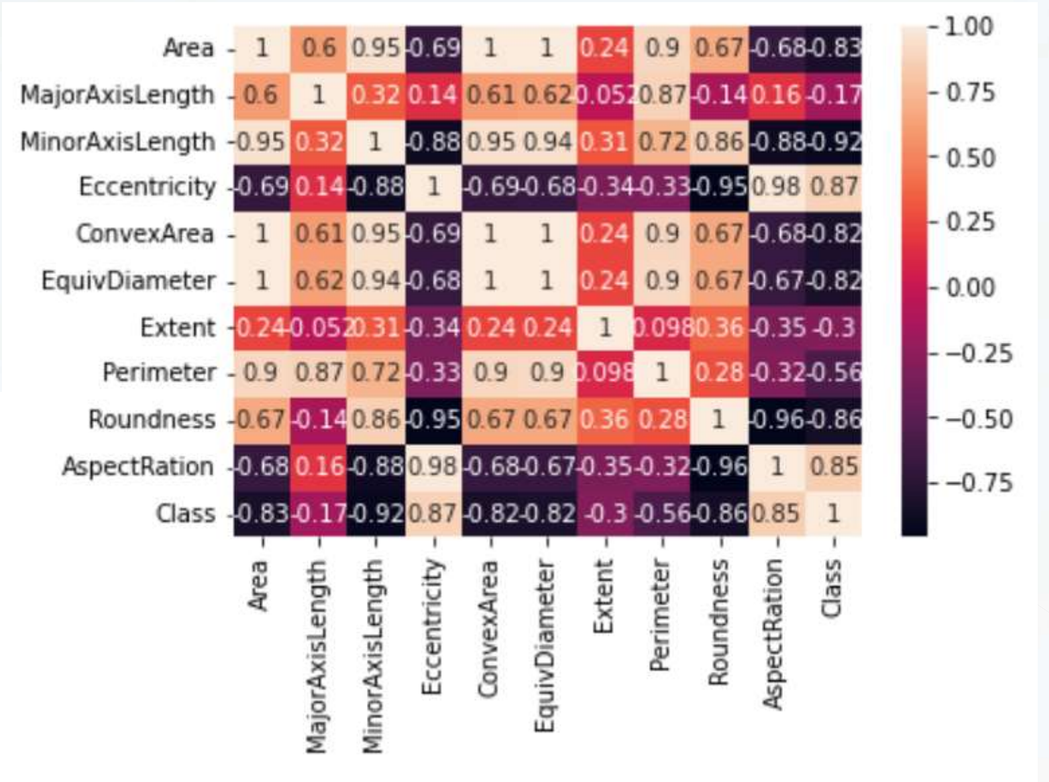
# АНАЛИЗ ДАННЫХ

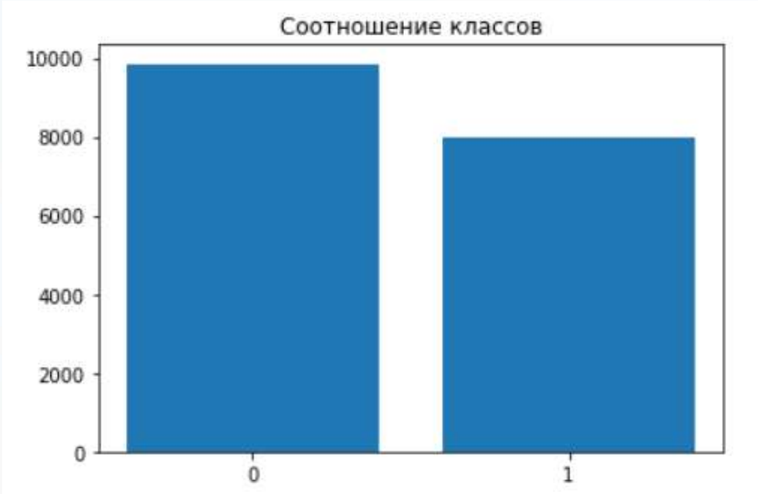
Синий – тип риса Gonen,
оранжевый – Jasmine

# АНАЛИЗ ДАННЫХ

Данные после удаления выбросов:



# ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

```
class LogisticRegression(ClassifierMixin, BaseEstimator):
    def __init__(self, epoches=1, batch_size=10, SGD_step=0.001, nin=10):
        self.epoches = epoches
        self.batch_size = batch_size
        self.SGD_step = SGD_step
        self.nin = nin
        self.Net = Net(BinaryCrossEntropy())
        self.Net.add(Linear(nin, 1))
        self.Net.add(Sigmoid())

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        self.X_ = X
        self.y_ = y
        self.is_fitted_ = True
        for _ in range(self.epoches):
            self.Net.train_epoch(X, y, self.batch_size, self.SGD_step)
        # Return the classifier
        return self

    def predict(self, X):
        check_is_fitted(self, ['X_', 'y_'])
        X = check_array(X)
        y = self.Net.forward(X)
        res = np.where(y < 0.5, 0, 1)
        return res

    def getW(self):
        return self.Net.layers[0].W

    def getb(self):
        return self.Net.layers[0].b
```

# РЕАЛИЗАЦИЯ СЕТИ

```
class Net:
    def __init__(self, loss=BinaryCrossEntropy()):
        self.layers = []
        self.loss_ = loss

    def add(self, l: Layer):
        self.layers.append(l)

    def forward(self, x):
        for l in self.layers:
            x = l.forward(x)
        return x

    def backward(self, z):
        for l in self.layers[::-1]:
            z = l.backward(z)
        return z

    def update(self, lr):
        for l in self.layers:
            if 'update' in l.__dir__():
                l.update(lr)

    def get_loss_acc(self, x, y):
        p = self.forward(x)
        l = self.loss_.forward(p, y)
        pred = np.argmax(p, axis=1)
        acc = (pred == y).mean()
        return l, acc

    def train_epoch(self, train_x, train_labels, batch_size=4, lr=0.1):
        for i in range(0, len(train_x), batch_size):
            xb = train_x[i:i + batch_size]
            yb = train_labels[i:i + batch_size]

            p = self.forward(xb)
            l = self.loss_.forward(p=p, y=yb)
            dp = self.loss_.backward(loss=l)
            dx = self.backward(z=dp)
            self.update(lr)
```

# СОДЕРЖИМОЕ СЕТИ

```python
class Layer:
    def forward(self, *args):
        pass

    def backward(self, *args):
        pass

class Linear(Layer):
    def __init__(self, nin, nout):
        sigma = 1.0 / np.sqrt(2.0 * nin)
        self.W = np.random.normal(0, sigma, (nout, nin))
        self.b = np.zeros((1, nout))
        self.dW = np.zeros_like(self.W)
        self.db = np.zeros_like(self.b)

    def forward(self, x):
        self.x = x
        return np.dot(x, self.W.T) + self.b

    def backward(self, dz):
        dx = np.dot(dz, self.W)
        dW = np.dot(dz.T, self.x)
        db = dz.sum(axis=0)
        self.dW = dW
        self.db = db
        return dx

    def update(self, lr):
        self.W -= lr * self.dW
        self.b -= lr * self.db
```

```python
class BinaryCrossEntropy(Layer):
    def forward(self, p, y):
        y = y.reshape((y.shape[0], 1))
        self.p = p
        self.y = y
        res = y * np.log(p) + (1 - y) * np.log(1 - p)
        return -np.mean(res)

    def backward(self, loss):
        res = (self.p - self.y) / (self.p * (1 - self.p))
        return res / self.p.shape[0]
```

```python
class Sigmoid(Layer):
    def forward(self, x):
        self.y = 1.0 / (1.0 + np.exp(-x))
        return self.y

    def backward(self, dy):
        return self.y * (1.0 - self.y) * dy
```

# ЛУЧШИЕ ПАРАМЕТРЫ

```
gscv = GridSearchCV(Pipeline([("logreg", LogisticRegression())]),
                    {"logreg__epoches" : [1, 2, 4],
                     "logreg__batch_size" : [5, 10, 20],
                     "logreg__SGD_step" : [0.01, 0.05, 0.1]})
gscv.fit(train_X, train_y)
```

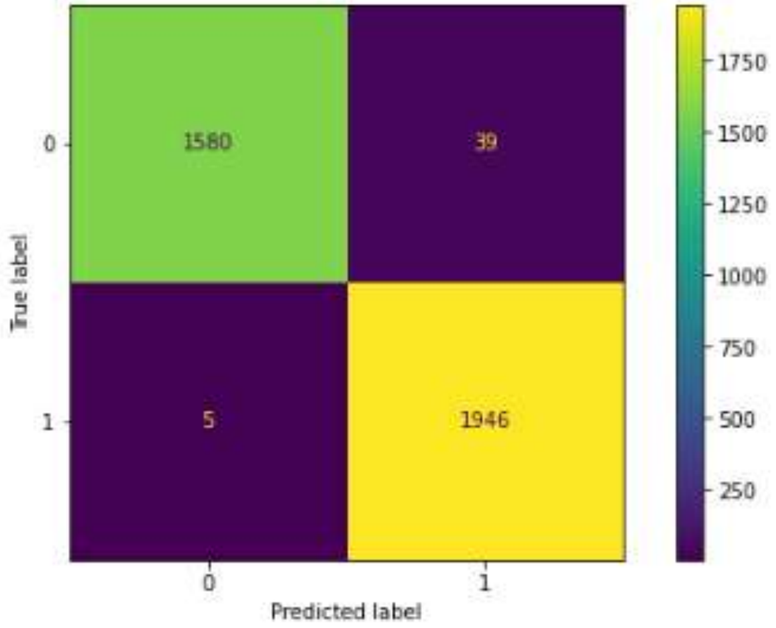
Лучшие параметры модели для данного датасета:

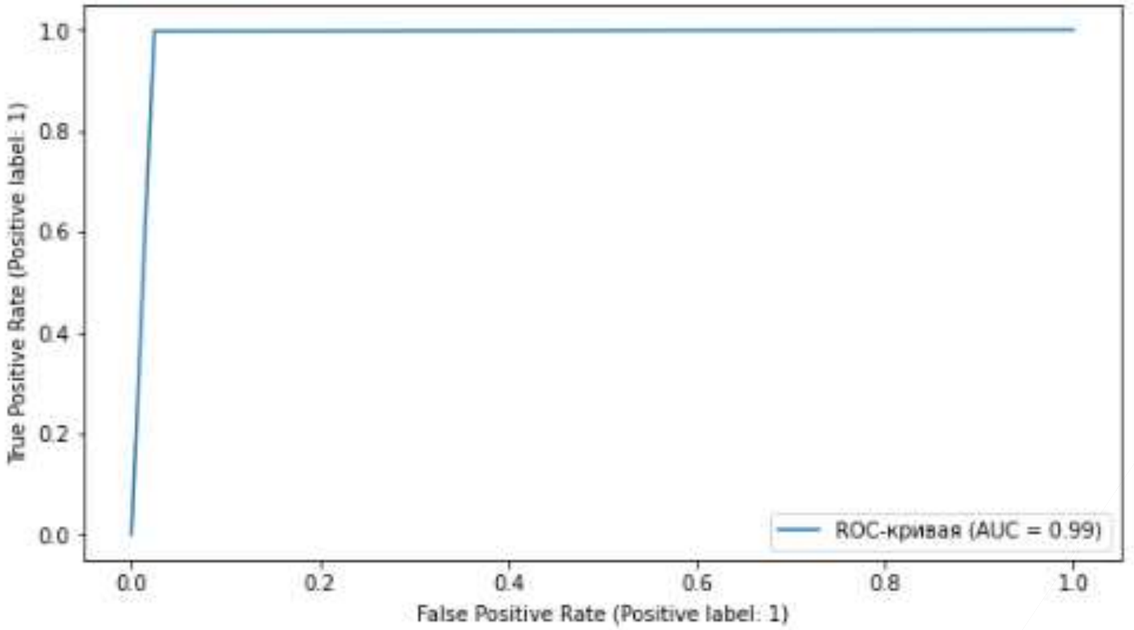
```
params: {'logreg__SGD_step': 0.1, 'logreg__batch_size': 5, 'logreg__epoches': 4}
```

# РЕЗУЛЬТАТ ОБУЧЕНИЯ

```
logreg_best = gscv.best_estimator_
scores(logreg_best, test_X, test_y)
```

```
Accuracy: 0.9876750700280112
Recall: 0.9974372116863147
Precision: 0.980352644836272
```



# QR-КОД НА РЕПОЗИТОРИЙ ПРОЕКТА