


## № 2.2 (Вариант-1)

Прежде чем развернуть среду программирования на языке Julia с использованием интегрированной среды разработки Visual Studio Code, необходимо установить ЯП julia:

```
brew install --cask julia
```

(в случае MacOS) и скачать расширение для VS Code, после чего в терминале стоит выполнить julia (что б убедиться что язык julia установился и готов к использованию):



```
Last login: Fri Sep 19 21:04:30 on ttys001
fedorcagin@MacBook-Pro13 ~ % julia

  _       _       _
 _/ _\   _/ _\   _/ _\   _/ _\   _/ _\   _/ _\   _/ _\
|  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |
|  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|
|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Documentation: https://docs.julialang.org
Type "?" for help, "]"? for Pkg help.
Version 1.11.7 (2025-09-08)
Official https://julialang.org/ release

julia>
```

При успешной установке следующее действие — создание Jupyter Notebook (файл с расширением .ipynb) в VSCode и выбор ядра Julia для работы. Для установки пакетов файле нужно ввести

```
using Pkg
```

```
Pkg.add(["RDatasets", "DataFrames", "MLJ", "MLJModels", "PrettyPrinting"])
```

,где

- RDatasets: предоставляет доступ к классическим наборам данных (включая Iris).
- DataFrames: для работы с табличными данными.
- MLJ: фреймворк для машинного обучения (аналог Scikit-Learn в Python).
- MLJModels: каталог моделей, совместимых с MLJ (указывает, где искать конкретные алгоритмы).
- PrettyPrinting: для удобного форматированного вывода результатов.

После успешной установки всех пакетов можно перейти к коду:

```
using RDatasets, DataFrames
```

```
using MLJ, MLJModels, PrettyPrinting
```

```
import MLJ: fit!, predict, machine
```

```
# Загрузка данных
```

```
iris = dataset("datasets", "iris")
```

```
first(iris, 5)
```

```
# Преобразует названия и пр. данные в числовые категории, которые понимает модель
```

```
y_coerce = coerce(iris.Species, Multiclass)
```

```
unique(y_coerce)
```

```
# Разделение данных
```

```
X = iris[:, 1:4]
```

```
y = y_coerce
```

```
# Разделение на общую и тестовую выборку
```

```
train, test = partition(eachindex(y), 0.8, shuffle=true, rng=42)
```

```
# Загрузка KNN модели
```

```
KNNClassifier = @load KNNClassifier
```

```
model = KNNClassifier()
```

```
# Создание и обучение модели
```

```
mach = machine(model, X, y)
```

```
fit!(mach, rows=train)
```

```
# Предсказания и оценка точности
```

```
y_pred = predict_mode(mach, rows=test) # (predict_mode сразу возвращает классы)
```

```
accuracy = mean(y_pred .== y[test])
```

```
println("Точность модели KNN: ", round(accuracy * 100; digits=2), "%")
```

```
import NearestNeighborModels ✓
Точность модели KNN: 96.67
└ Info: For silent loading, specify `verbosity=0`.
└ @ Main /Users/fedorcagin/.julia/packages/MLJModels/kHs85/src/loading.jl:159
└ Info: Training machine(KNNClassifier(K = 5, ...), ...).
└ @ MLJBase /Users/fedorcagin/.julia/packages/MLJBase/F8Zzu/src/machines.jl:499
%
```

Результат:

Код берёт значения 120 цветков из таблицы, говорит модели что это за цветы (выполняя обучающую функцию), а потом берёт 30 и отдаёт их в MLJ уже на тестирование. Точность модели KNN составила 96.67% таким образом ошибка произошла всего в 1 случаев.