МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра ИС

ОТЧЁТ

по практической работе № 6

по дисциплине «Статистический анализ»

Тема: Кластерный анализ. Метод k-средних

Вариант № 8

Студент гр. 9372 Иванов Р.С.

Преподаватель Сучков А.И.

Санкт-Петербург

2021

Цель работы.

Освоение основных понятий и некоторых методов кластерного анализа, в частности, метода k-means.

Основные теоретические положения.

Формулы для формирования интервального вариационного ряда:

$k = \lfloor 1 + \ln(n) \rfloor$ - количество интервалов, где n - объем выборки.

R = xmax - xmin - размах выборки

$h = \frac{R}{k}$ - длина интервала

$\tilde{x}_i = \frac{x_i - \bar{x}_B}{\sigma_B}$ – нормализация

Алгоритм k-means

Шаги алгоритма:

1. **Начальный шаг: инициализация кластеров**
   Выбирается произвольное множество точек μ_i, $i=1,...,k$, рассматриваемых как начальные центры кластеров: $\mu(0)_i=\mu_i$, $i=1,...,k$
2. **Распределение векторов по кластерам**
   Шаг t: $\forall x_i \in X, i=1...n : x_i \in S_j \Leftrightarrow j = \operatorname{argmin}_k \rho(x_i,\mu^{(t-1)}_k)^2$
3. **Пересчет центров кластеров**
   Шаг t: $\forall I =1,...,k : \mu^{(t)}_i=\frac{1}{|S_i|}\sum_{x \in S_i} x$
4. **Проверка условия останова:**
   - **if** $\exists i \in \overline{1,k} : \mu^{(t)}_i \neq \mu^{(t-1)}_i$ **then**
     - $t=t+1$;
     - goto 2;
   - **else**
     - **stop**

Постановка задачи.

Дано конечное множество из объектов, представленных двумя признаками (в качестве этого множества принимаем исходную двумерную выборку, сформированную ранее в практической работе №4). Выполнить разбиение исходного множества объектов на конечное число подмножеств (кластеров) с использованием метода k-means. Полученные результаты содержательно проинтерпретировать.

Выполнение работы.

Во время выполнения работы был написан код на языке Python, выполняющий поставленную задачу. Реализовано чтение и дальнейшая обработка данных из Price_Mileage.csv, одобренного преподавателем. Для этого была использована библиотека csv.

Пункт 1

Нормализовано множество точек из предыдущего раздела, на графике отображено полученное множество.

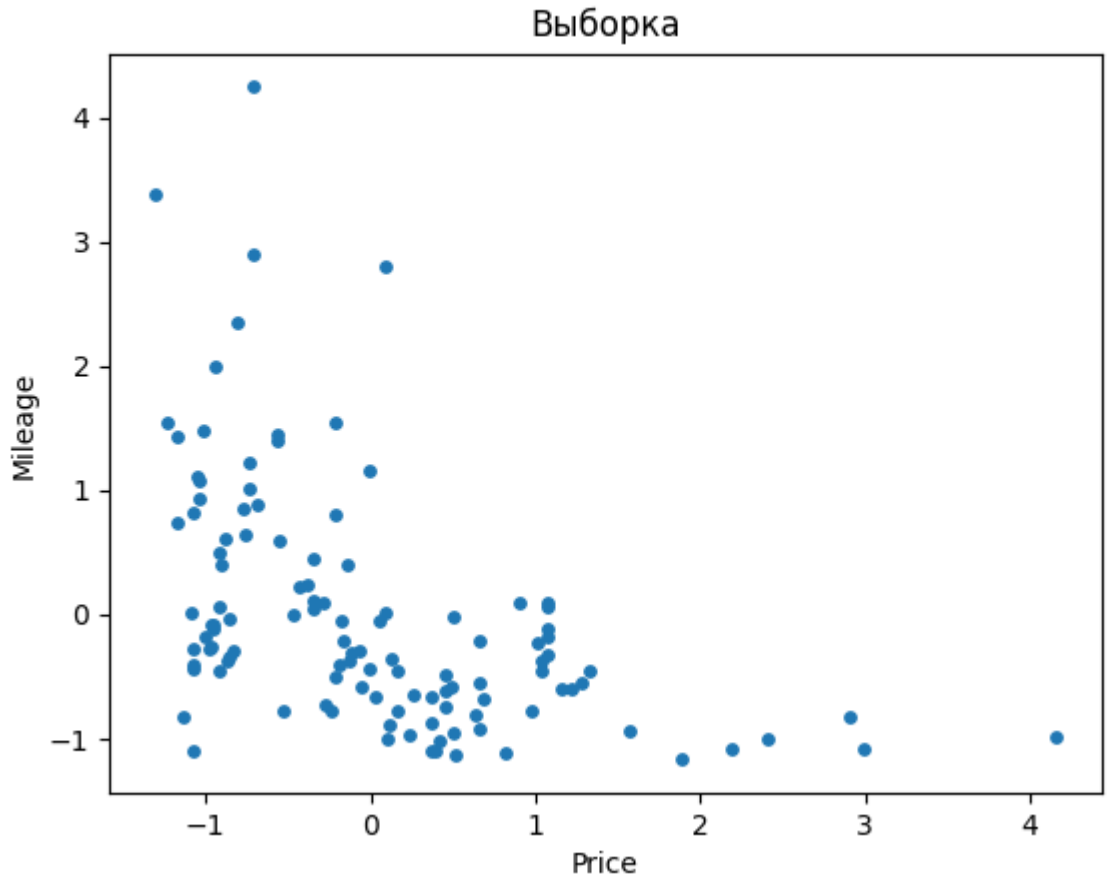

Рисунок 1 (Выборка)

Пункт 2

Определена грубая верхняя оценка количества кластеров.

$$k = 1 + \lfloor \ln(115) \rfloor = 8$$

Пункт 3

Реализован алгоритм k-means. Отображены полученные кластеры, выделен каждый кластер разным цветом, отмечены центроиды.

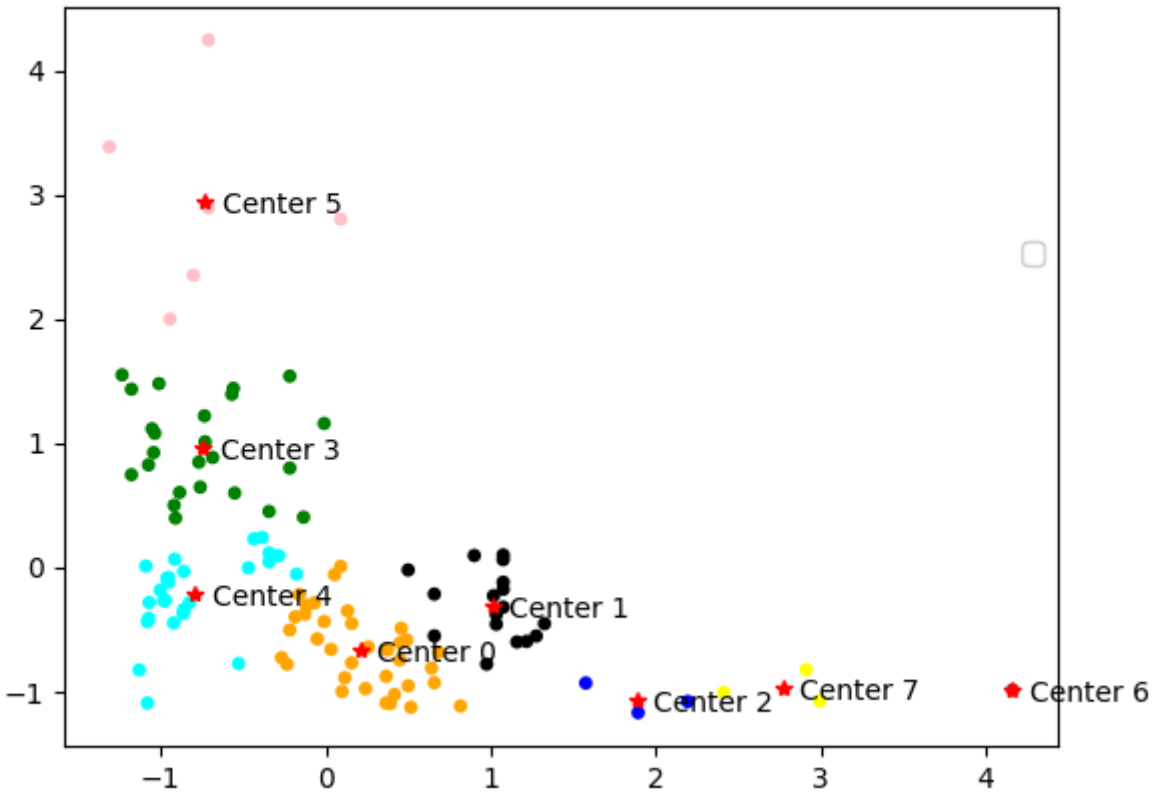

Рисунок 2 (K-means)

Пункт 4

Реализован алгоритм kmedoids (для чётных вариантов). Отображены полученные кластеры, выделен каждый кластер разным цветом, отмечены центроиды.

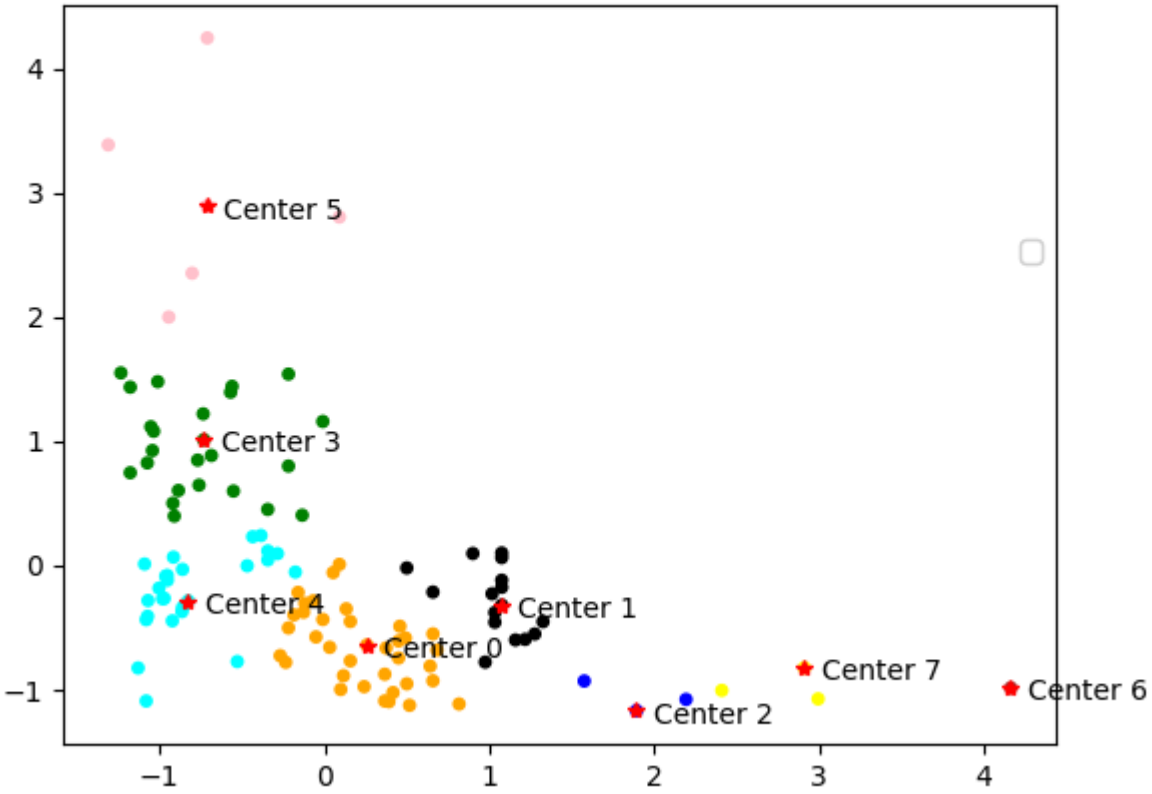

Рисунок 3 (K-medoids)

Пункт 5

Построен график для определения оптимального количества кластеров с помощью метода силуэтов.

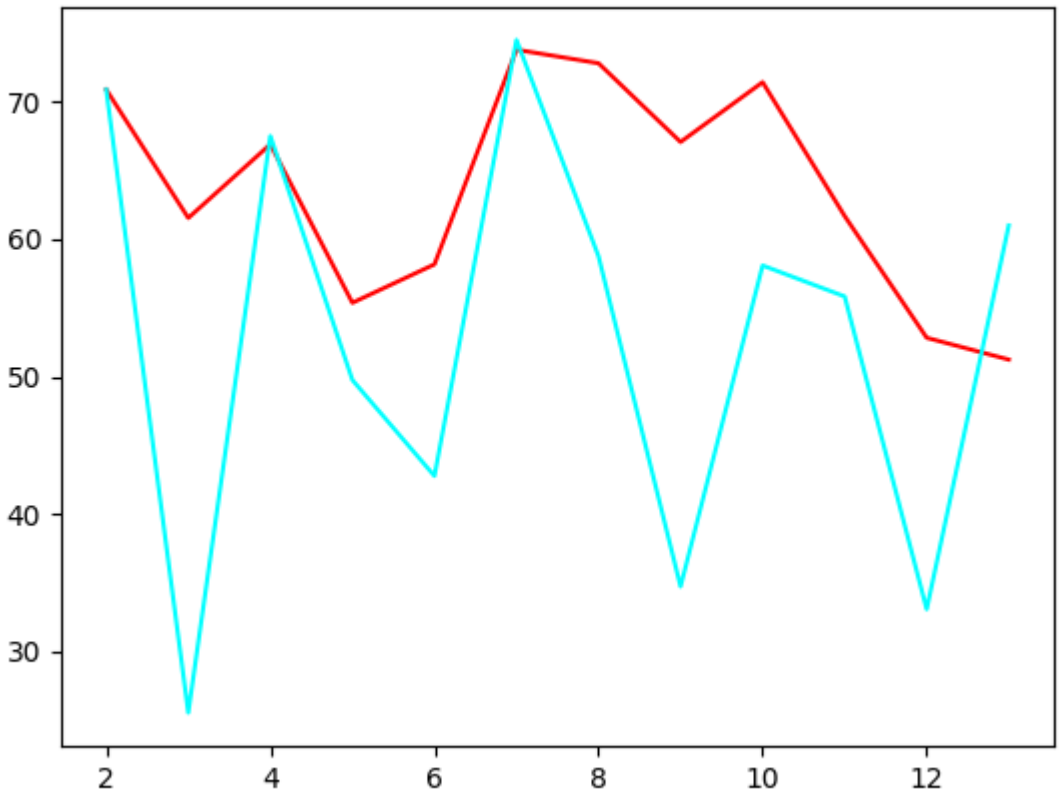

Рисунок 4 (Метод силуэтов)

Красная линия на графике — это применение метода силуэтов для k-means, синяя – k-medoids. По графику можно сказать, что для обоих методов оптимальным количеством кластеров является k = 7.

Пункт 6

Были оценены методы разбиения с помощью получения суммарного внутрикластерного расстояния между точками. Были получены следующие величины:

K-means	K-medoids
21.94	35.31

По полученным данным можно сказать, что на данной выборке лучше работает алгоритм k-means. После его применение внутрикластерное расстояние меньше, чем при применении алгоритма k-medoids.

Пункт 7

Алгоритмы k-means и k-medoids очень похожи по своей реализации, единственное отличие отличается в том методе пересчёта центройдов, в одном случае это точка(не обязательно присутствующая в кластере) расстояние до которой минимально для каждого члена кластера, в другом обязательно лежащая в кластере точка.

Вывод.

В процессе выполнения практической работы были реализованы такие алгоритмы разбиения на кластеры, как k-means , k-medians. Выполнена верхняя оценка количества кластеров и оценка оптимального количества кластеров методом локтя. Заметили, что в алгоритмах k-means и k-medians первичные центроиды выбираются случайно из множества всех точек, таким образом, эти алгоритмы очень чувствительны к их выбору.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import numpy as np
import matplotlib.pyplot as plt
import math
import csv
# import scipy
import random
import sklearn
from openpyxl import Workbook

# from openpyxl.drawing.image import Image
from sklearn.cluster import KMeans


def calc_distance(coordinate_1, coordinate_2):
    return (coordinate_1[0] - coordinate_2[0]) ** 2 + (coordinate_1[1] -
coordinate_2[1]) ** 2


def recalc_centroids(cluster):
    centroid = []
    temp_1 = 0
    temp_2 = 0
    len_cluster = len(cluster)
    for i in range(len_cluster):
        temp_1 += cluster[i][0]
        temp_2 += cluster[i][1]
    centroid.append(temp_1 / len_cluster)
    centroid.append(temp_2 / len_cluster)
    return centroid


def recalc_medoids(s, clu, medoid):
    m = 10 ** 10
    ans = -1
    for i in range(len(clu)):
        med = medoid
        cluL = clu
        med = s[cluL[i]]
        rang = all_ranges(s, clu, med)
        if rang < m:
            m = rang
            ans = i
    if ans != -1:
        medoid = s[clu[ans]]
    return clu, medoid


def all_ranges(s, clu, medoid):
    ans = 0
    for i in range(len(clu)):
        ans += calc_distance(medoid, s[clu[i]])
    return ans
```

```
def calc_distance(coordinate_1, coordinate_2):
    return (coordinate_1[0] - coordinate_2[0]) ** 2 + (coordinate_1[1] -
coordinate_2[1]) ** 2


def cluster_index(D, cluster):
    return [D.index(coordinate) for coordinate in cluster]


def k_means(centers, s):
    centers_new = centers
    ku = len(centers)
    for max in range(200):
        thr = []
        clu = []
        clus = []
        for i in range(ku):
            clus.append([])

        for point in s:
            distance = []
            min = 1000
            l = 0
            for j in range(ku):
                distance.append(calc_distance(centers_new[j], point))

                if distance[j] < min:
                    min = distance[j]
                    l = j
            clus[l].append(point)
        centers_old = centers_new.copy()
        for i in range(ku):
            centers_new[i] = recalc_centroids(clus[i])
        for j in range(ku):
            thr.append(bool(calc_distance(centers_old[j], centers_new[j])))
        for l in range(ku):
            clu.append(cluster_index(s, clus[l]))
        if not any(thr):
            break
    return centers_new, clu


def k_meds(meds, s):
    med_new = meds
    ku = len(meds)
    for mx in range(200):
        thr = []
        clu = []
        clus = []
        for i in range(ku):
            clus.append([])

        for point in s:
            distance = []
            min = 1000
            l = 0
            for j in range(ku):
                distance.append(calc_distance(med_new[j], point))
```

```
                if distance[j] < min:
                    min = distance[j]
                    l = j
            clus[l].append(point)
        meds_old = med_new.copy()
        for l in range(ku):
            clu.append(cluster_index(s, clus[l]))
        for i in range(ku):
            clu[i], med_new[i] = recalc_medoids(s, clu[i], meds[i])
        for j in range(ku):
            thr.append(calc_distance(meds_old[j], med_new[j]))
        if not any(thr):
            break
    return med_new, clu


def make_plot(centers, sample, clusters):
    mycolors = ['orange', 'black', 'blue', 'green', 'cyan', 'pink', 'brown', 'yellow']
    sample = np.array(sample)
    fig, ax = plt.subplots()
    for i, coordinate in enumerate(centers):
        ax.plot(coordinate[0], coordinate[1], 'r*')
        ax.text(coordinate[0] + 0.1, coordinate[1] - 0.1, 'Center ' + str(i))
    labels_ = [0] * len(sample)
    sum =0
    for i in range(len(clusters)):
        for index in range(len(clusters[i])):
            labels_[clusters[i][index]] = mycolors[i]

    # for cluster in clusters:
    #     for index in cluster:
    #         ax.text(D[index, 0] + 0.01, D[index, 1] + 0.1, index)
    scatter = ax.scatter(sample[:, 0], sample[:, 1], s=15, c=labels_, cmap='rainbow')
    legend = ax.legend(*scatter.legend_elements(fmt='Cluster {x:.0f}'),
bbox_to_anchor=(1, 0.7))
    ax.add_artist(legend)
    plt.show()


sample = []

with open('Price-Mileage.csv') as csv_file:  # Читаем выборку из файла 0 работы
    spam_reader = csv.reader(csv_file, quotechar='|')
    for row in spam_reader:
        x, y = row[0].split(';')
        if y.isdigit() and x.isdigit():
            sample.append([int(x), int(y)])

n = len(sample)
k = math.ceil(math.sqrt(n / 2))

Xs = 0  # Выброчное среднее по x
Ys = 0  # По y

Dsx = 0  # Дисперсия по x
Dsy = 0  # По s

sy = 0  # Исправленное СКО по y
```

```
sx = 0  # По x

for i in range(n):
    Xs += sample[i][0]
Xs = Xs / n

for i in range(n):
    Ys += sample[i][1]
Ys = Ys / n

for i in range(n):
    Dsx += (sample[i][0] - Xs) ** 2
Dsx = Dsx / n

for i in range(n):
    Dsy += (sample[i][1] - Ys) ** 2
Dsy = Dsy / n

sx = math.sqrt(Dsx) * n / (n - 1)
sy = math.sqrt(Dsy) * n / (n - 1)

for i in range(n):  # Нормализация точек
    sample[i][0] = (sample[i][0] - Xs) / sx
    sample[i][1] = (sample[i][1] - Ys) / sy

X = [sample[i][0] for i in range(n)]
Y = [sample[i][1] for i in range(n)]

plt.scatter(x=X, y=Y, s=15)
ax = plt.gca()
ax.set(title='Выборка')
#  Добавляем подписи к осям:
ax.set_xlabel('Price')
ax.set_ylabel('Mileage')
plt.show()

e = 0.01
means_init = random.sample(sample, k)

U, clusters = k_means(means_init, sample)
make_plot(U, sample, clusters)

meds_init = random.choices(sample, k = k)
meds_index = []
# for i in range(k):
#     for j in range(len(sample)):
#         if meds_init[i] == sample[j]:
#             meds_index.append(j)
#             break
# samp = sample
# samp = [x for x in samp if x not in meds_init]

meds_init, clusters = k_meds(means_init, sample)
make_plot(meds_init, sample, clusters)

def silhouette(centers, clus, samp):
    for i in range(len(samp)):
        for j in range(len(clus)):
```

```
            for l in range(len(clus[j])):
                if i == clus[j][l]:
                    samp[i].append(j)
                    break
            if len(samp[i]) == 3:
                break

    def min_rang(point, cens, sam):
        mi = 10*10
        for i in range(len(cens)):
            if i != point[2]:
                ze = calc_distance(point, cens[i])
                if ze < mi:
                    mi = ze
                    ans = i
        return(ans)

    ans = 0
    for i in range(len(samp)):
        a = all_ranges(samp, clus[samp[i][2]], samp[i])
        b = all_ranges(samp, clus[min_rang(samp[i], centers, samp)], samp[i])
        ans += (b-a)/max(a, b)
        samp[i].pop(2)
    return ans

silhous_means = []
silhous_meds = []
for i in range(2, 14):
    means_init = random.sample(sample, i)
    means_init, clusters1 = k_means(means_init, sample)
    silhous_means.append(silhouette(means_init, clusters1, sample))


    meds_init = random.choices(sample, k=i)
    meds_init, clusters2 = k_meds(meds_init, sample)
    silhous_meds.append(silhouette(meds_init, clusters2, sample))

    #make_plot(means_init, sample, clusters1)
    #make_plot(meds_init, sample, clusters2)

X = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
plt.plot(X, silhous_means, color='red')
plt.plot(X, silhous_meds, color='cyan')
plt.show()

meds_init = random.choices(sample, k=7)
meds_init, clusters2 = k_meds(meds_init, sample)

means_init = random.sample(sample, 7)
means_init, clusters1 = k_means(means_init, sample)

def ranges_clusters(clus, centrs, samp):
    ans = 0
    for i in range(len(clus)):
        ans += all_ranges(samp,clus[i],centrs[i])
    return ans

qual_mean = ranges_clusters(clusters1, means_init, sample)
```

```
qual_meds = ranges_clusters(clusters2, meds_init, sample)

make_plot(means_init, sample, clusters1)
make_plot(meds_init, sample, clusters2)

print(qual_mean)
print(qual_meds)
```