МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет «Высшая школа экономики»
Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

ОТЧЕТ
О ВЫПОЛНЕНИИ ИТОГОВОЙ РАБОТЫ ПО КУРСУ
CORE CONCEPCTS IN DATA ANALYSIS

Выполнили	*Студент 3 курса ПМИ ФКН ВШЭ, группы БПМИ174*	Никифоров Алексей Владимирович
	Студент 3 курса ПМИ ФКН ВШЭ, группы БПМИ174	Попов Илья Иванович
Руководитель	*Профессор департамента анализа данных и искусственного интеллекта, доктор технических наук*	Миркин Борис Григорьевич

Москва, 2019

СОДЕРЖАНИЕ

ВЫБОР И ОПИСАНИЕ ДАННЫХ

Для выполнения работы мы использовали исторические данные о ноябрьской погоде в Москве в период с 1999 по 2014 год включительно. Данные были взяты с сайта *climate-energy.ru*[1], где автор публикует архивы погоды по Москве, городам Московской области и некоторым городам России. Ниже приводится пример данных из датасета, а также описание признаков.

date	avg_temp	min_temp	max_temp	downfall	pressure	humidity	wind_speed	snow_depth	year
1	1.3	-1.0	2.7	0.0	1020.7	80.0	1.7	0	2014
2	-1.7	-3.0	-0.4	0.0	1026.2	74.0	1.9	0	2014
3	-0.4	-1.0	0.5	5.6	1020.0	89.0	1.0	0	2014
5	3.2	2.0	4.4	0.0	1014.6	92.0	1.0	0	2014
6	5.1	4.0	5.8	0.0	1018.7	95.0	1.0	0	2014
7	5.8	4.0	7.3	4.1	1016.7	97.0	1.0	0	2014
8	7.4	6.0	8.6	0.0	1014.0	88.0	1.1	0	2014
9	4.2	3.0	5.8	5.1	1024.0	86.0	1.0	0	2014
11	4.6	3.0	5.3	0.0	1030.3	92.0	1.2	0	2014
12	5.4	5.0	5.7	0.0	1026.0	82.0	1.0	0	2014

Таблица 1 - Пример данных из датасета

Описание признаков в датасете:

- `date` – целое число – номер для в месяце;
- `avg_temp` – действительное число – средняя температура в течении дня (в градусах Цельсия);
- `min_temp` – действительное число – минимальная температура в течении дня (в градусах Цельсия);
- `max_temp` – действительное число – максимальная температура в течении дня (в градусах Цельсия);
- `downfall` – действительное число – количество осадков в течении дня (в миллиметрах на один квадратный метр);
- `pressure` – действительное число – среднее атмосферное давление в течении дня (в миллиметрах ртутного столба);
- `humidity` – действительное число – средняя относительная влажность воздуха в течении дня (в процентах);
- `wind_speed` – действительное число – средняя скорость ветра в течении дня (в метрах в секунду);

- `snow_depth` – действительное число – глубина снежного покрова (в сантиметрах);
- `year` – целое число – год сбора данных;

Итого, в нашем датасете имеется 375 записей и 10 признаков. Ссылку на файл в формате CSV со всеми данными можно найти в Списке использованных материалов[2]. Работа выполнялась на языке Python, с использованием среды Google Colab[3] для осуществления совместной работы над проектом, а также с использованием инструментов из библиотек Pandas[4], NumPy[5], Matplotlib[6] и SKLearn[7].

КЛАСТЕР-АНАЛИЗ

Для выполнения задания по реализации кластерного анализа из датасета были выбраны следующие количественные признаки:

- `avg_temp` – средняя температура в течении дня (в градусах Цельсия);
- `pressure` – среднее атмосферное давление в течении дня (в миллиметрах ртутного столба);
- `wind_speed` – средняя скорость ветра в течении дня (в метрах в секунду);

Именно эти признаки были выбраны для проведения анализа потому, что они не коррелируют между собой напрямую, и, как нам кажется, позволяют наилучшим образом определить группы схожих дней.

Для выполнения непосредственно анализа был использован модуль KMeans[8] библиотеки SKLearn[7]. Рассмотрим код реализации анализа K-средних для 5 кластеров:

```
1.  from sklearn.cluster import KMeans
2.
3.  # Выбираем нужные нам признаки из общего датасета
4.  df_features = df[['avg_temp', 'pressure', 'wind_speed']]
5.
6.  kmeans_5 = KMeans(
7.      n_clusters = 5,       # 5 кластеров
8.      init = 'random',      # центры кластеров - случайно выбранные точки из датасета
9.      n_init = 10           # количество инициализаций алгоритма
10. )
11. kmeans_5.fit(df_features) # выполнение алгоритма
```

В настройках метода были указаны необходимые параметры работы алгоритма:

- `n_clusters=5` – количество кластеров
- `init='random'` – в качестве центров кластеров при инициализации алгоритма используются случайные объекты из датасета
- `n_init=10` – метод производит 10 случайных инициализаций алгоритма и выбирает то, которое удовлетворяет минимуму критерия метода;

Рассмотрим также визуализацию полученных кластеров, реализованную с помощью методов библиотеки Matplotlib[6] (полный исходный код слишком большой, он приведен ниже в приложении).

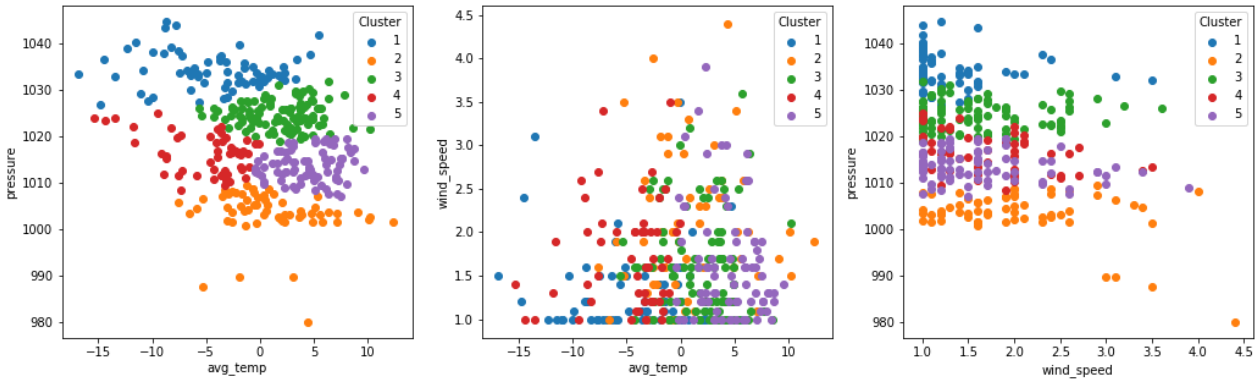

Иллюстрация 1 - Визуализация кластеров, полученных методов KMeans для k=5

Теперь выполним аналогичные действия для k=9:

```
from sklearn.cluster import KMeans

# Выбираем нужные нам признаки из общего датасета
df_features = df[['avg_temp', 'pressure', 'wind_speed']]

kmeans_9 = KMeans(
    n_clusters = 9,       # 9 кластеров
    init = 'random',      # центры кластеров - случайно выбранные точки из датасета
    n_init = 10           # количество инициализаций алгоритма
)
kmeans_9.fit(df_features) # выполнение алгоритма
```

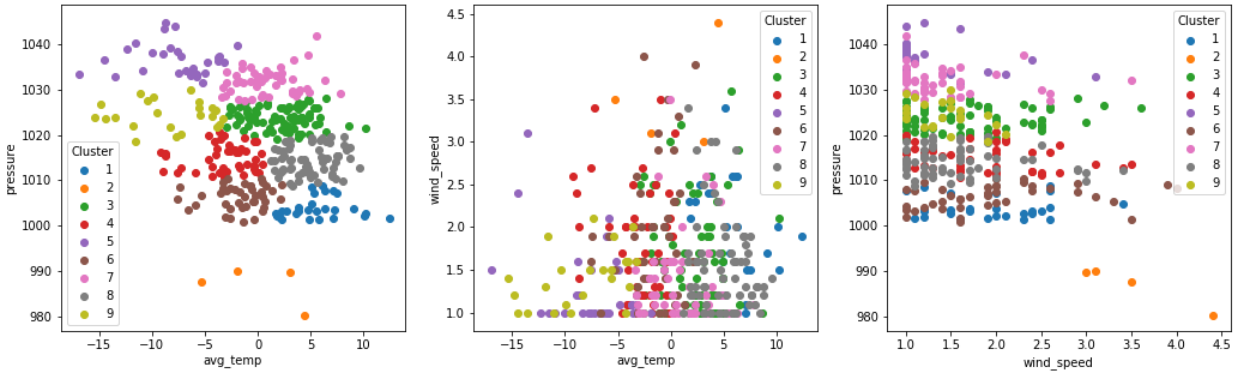

Иллюстрация 2 - Визуализация кластеров, полученных методов KMeans для k=9

Проинтерпретируем оба разбиения с помощью признаков таблицы данных путем сравнения внутрикластерных средних с общими средними. Для этого сделаем следующие визуализации:

cluster	avg_temperature	avg_pressure	avg_wind
All data	0.186376	1019.381471	1.645504
Cluster #0	5.559259	1003.914815	1.929630
Cluster #1	0.075000	986.850000	3.500000
Cluster #2	2.156627	1023.771084	1.603614
Cluster #3	-3.089130	1015.043478	1.819565
Cluster #4	-7.672414	1036.734483	1.310345

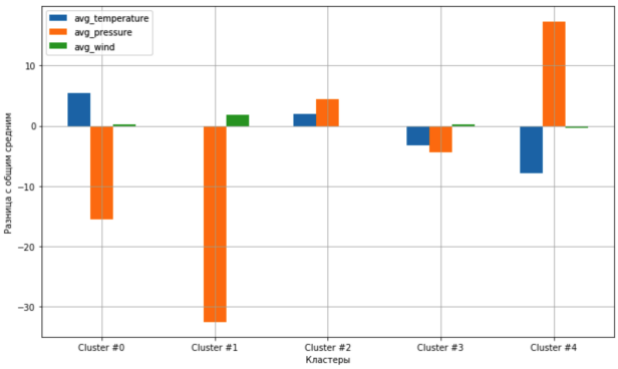

Иллюстрация 3 - Таблица средних и bar chart разницы внутрикластерного и общего средних для разбиения на 5 кластеров

Хорошо видно, что в Кластер №0 попали дни с температурой выше средней и давлением ниже среднего, в Кластер №1 – дни с аномально низким атмосферным давлением, а в Кластер №4 – дни с температурой ниже среднего и атмосферным давлением сильно выше среднего (что в народе называется «мороз и солнце»). Для остальных кластеров внутрикластерные средние значение признаков не сильно отличаются от средних по всему датасету.

cluster	avg_temperature	avg_pressure	avg_wind
All data	0.186376	1019.381471	1.645504
Cluster #0	5.559259	1003.914815	1.929630
Cluster #1	0.075000	986.850000	3.500000
Cluster #2	2.156627	1023.771084	1.603614
Cluster #3	-3.089130	1015.043478	1.819565
Cluster #4	-7.672414	1036.734483	1.310345
Cluster #5	-1.665854	1006.353659	1.965854
Cluster #6	0.952000	1031.764000	1.380000
Cluster #7	4.943750	1014.351562	1.575000
Cluster #8	-8.352174	1024.452174	1.417391

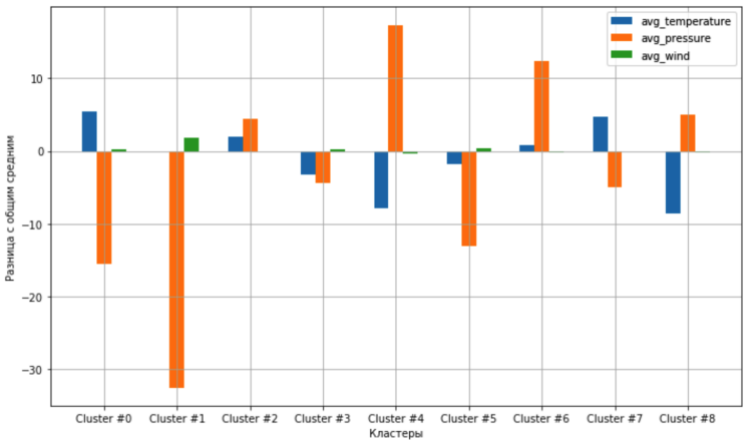

Иллюстрация 4 - Таблица средних и bar chart разницы внутрикластерного и общего средних для разбиения на 9 кластеров

Похожую картину мы наблюдаем и для разбиения на 9 кластеров:

- в Кластер №0 попали дни с температурой выше средней и давлением ниже среднего
- в Кластер №1 попали дни с давлением сильно ниже среднего и скоростью ветра сильно выше среднего
- в Кластер №4 попали дни с температурой ниже средней и аномально высоким давлением («мороз и солнце»)
- в Кластер №5 попали дни с температурой и давлением ниже среднего
- в Кластер №6 попали дни с температурой и давлением выше среднего
- для остальных кластеров можно сказать, что их внутрикластерные средние близки к средним на все датасете

Глядя на оба разбиения, нельзя точно сказать, какое из них лучше с точки зрения интерпретации. Оба разбиения имеют право на жизнь, и они хорошо подойдут для решения разных задач.

БУТСТРЭП

Найдем 95% доверительный интервал для среднего значения признака `avg_temp` на всем множестве объектов, используя бутстрэп:

```
from random import choices
from statistics import mean

# Список средних
means = []

# Количество экспериментов
bootstrap_iters = 5000
# Количество выбираемых значений для эксперимента
bootstrap_choices = len(df_features)

for i in range(bootstrap_iters):
    # Добавляем в список
    means.append(
        # Среднее значение
        mean(
            choices(
                # Выбираем случайный элемент из значений признака
                df_features_5['avg_temp'].tolist(),
                # k-раз
                k=bootstrap_choices
            )
        )
    )

# Визуализация
fig, graph = plt.subplots(figsize=(10, 6))
plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')
```

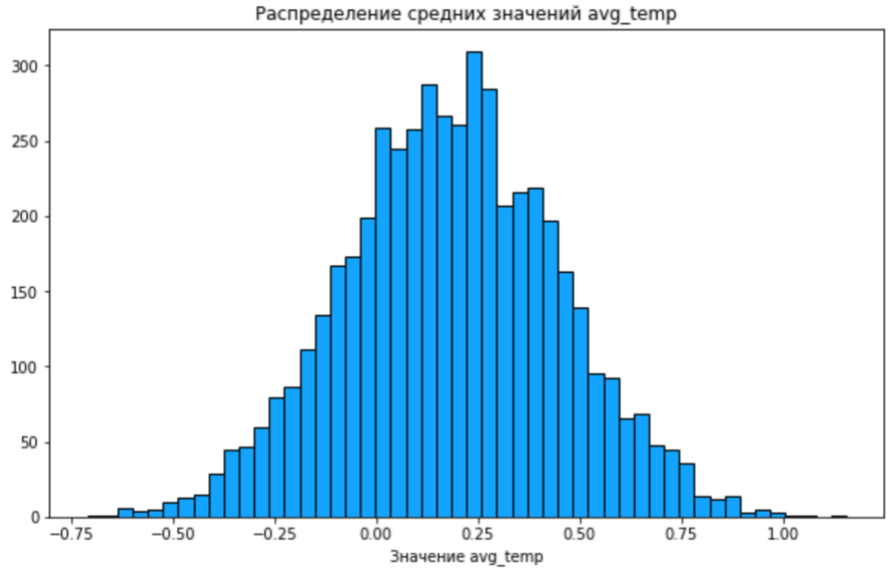

Иллюстрация 5 - Распределение средних значений avg_temp

Найдем 95% доверительный интервал способом "без опоры":

```
# Сортируем список со средними
means_ordered = sorted(means)
# Отсекаем 2.5% значений снизу и берем наименьший оставшийся
left = means_ordered[int(len(means_ordered)*0.05/2 - 1)]
# Отсекаем 2.5% значений сверху и берем наибольший оставшийся
right = means_ordered[int(len(means_ordered) - (len(means_ordered)*0.05/2) - 1)]

print(f'''''
  Границы 95% доверительного интервала:
  \tЛевая:\t{left}
  \tПравая:\t{right}
''')

>>>Границы 95% доверительного интервала:
>>>    Левая:  -0.34005449591280656
>>>    Правая: 0.6956403269754768
```

Найдем 95% доверительный интервал способом "с опорой":

```
from statistics import mean, stdev

print(f'''''
  Границы 95% доверительного интервала:
  \tЛевая:\t{mean(means) - 1.96*stdev(means)}
  \tПравая:\t{mean(means) + 1.96*stdev(means)}
''')

>>>Границы 95% доверительного интервала:
>>>    Левая:  -0.33938573245791703
>>>    Правая: 0.7020028441745383
```

Сравним средние по признаку avg_temp в кластерах №3 и №4 (из разбиения на 5 кластеров), используя бутстрэп. Для этого немного модифицируем приведенный выше код:

```
means = []
bootstrap_iters = 5000

for i in range(bootstrap_iters):
    means.append(
        # Из среднего значения признака в эксперименте на 3 кластере
        mean(
            choices(
                df_features_5[df_features_5.cluster == 3]['avg_temp'].tolist(),
                k=len(df_features_5[df_features_5.cluster == 3])
                )
            )
        # вычитаем
        -
        # среднее значение признака в эксперименте на 4 кластере
        mean(
            choices(
                df_features_5[df_features_5.cluster == 4]['avg_temp'].tolist(),
                k=len(df_features_5[df_features_5.cluster == 4])
                )
            )
    )

# Визуализация
fig, graph = plt.subplots(figsize=(10, 6))
plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')
```

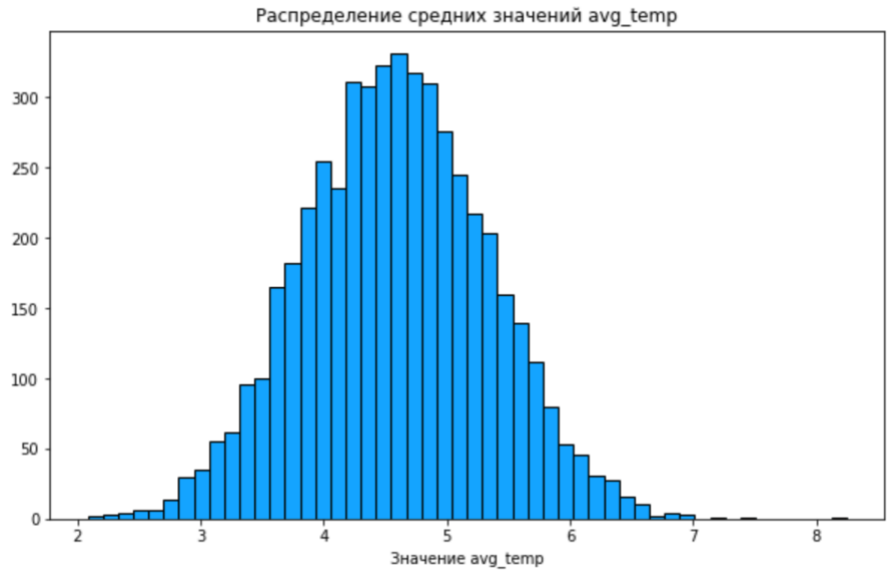

Иллюстрация 6 - Распределение разницы средних внутрикластерных значений признака avg_temp в кластерах №3 и №4

Ноль не попадает в это распределение, из чего можно сделать вывод, что средние значения признака `avg_temp` в кластерах №3 и №4 сильно различаются.

Рассмотрим распределения по-отдельности и рассчитаем 95% доверительный интервал для полученного выше распределения.

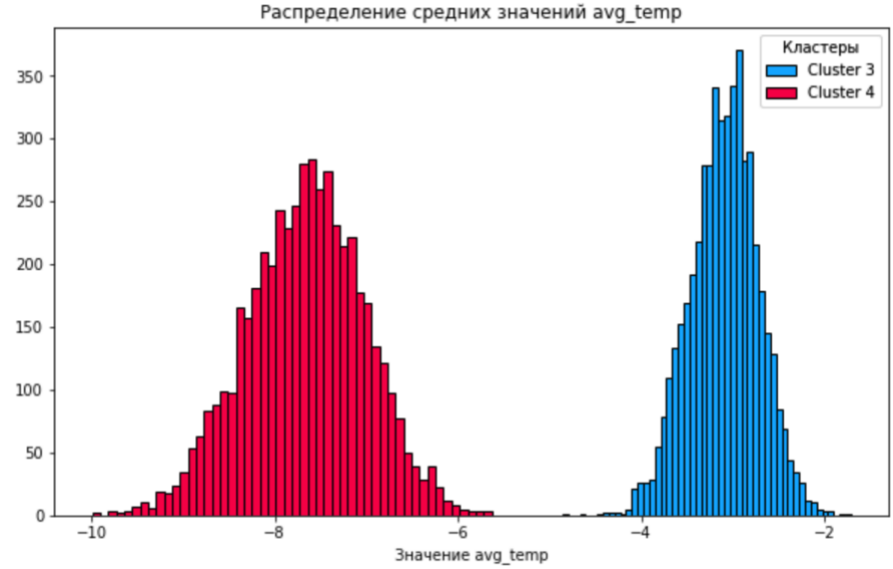

Иллюстрация 7 - Распределение средних внутрикластерных значений признака avg_temp в кластерах №3 и №4

Методика расчета доверительных интервалов абсолютно аналогична приведенной выше, поэтому ниже будут приведены только конечные результаты:

```
>>>Границы 95% доверительного интервала (без опоры):
>>>    Левая:  3.112893553223388
>>>    Правая: 6.068665667166417
>>>
>>>Границы 95% доверительного интервала (с опорой):
>>>    Левая:  3.1109550134891464
>>>    Правая: 6.043845406300958
```

Для кластера №1 (из разбиения на 5 кластеров) сравним среднее на всем множестве для признака `avg_temp` с его средним внутри кластера, используя бутстрэп:

```
means = []

bootstrap_iters = 5000

for i in range(bootstrap_iters):
    means.append(
        mean(
            choices(
                df_features_5[df_features_5.cluster == 1]['avg_temp'].tolist(),
                k=len(df_features_5[df_features_5.cluster == 1])
                )
            )
        -
        mean(
            choices(
                df_features_5['avg_temp'].tolist(),
                k=len(df_features_5)
                )
            )
    )

fig, graph = plt.subplots(figsize=(10, 6))
plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')
```

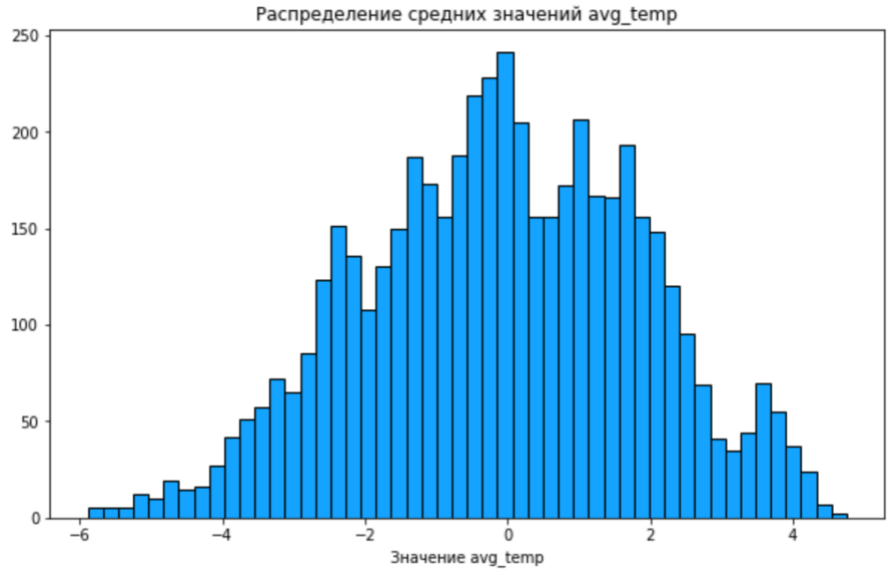

Иллюстрация 8 - Распределение разницы среднего внутрикластерного значения признака avg_temp в кластере №1 с среднем значением на всей выборке

Ноль попадает в это распределение, из чего можно сделать вывод, что средние значения признака `avg_temp` в кластере №1 и на всей выборке не сильно различаются.

Рассмотрим распределения по-отдельности и рассчитаем 95% доверительный интервал для полученного выше распределения.

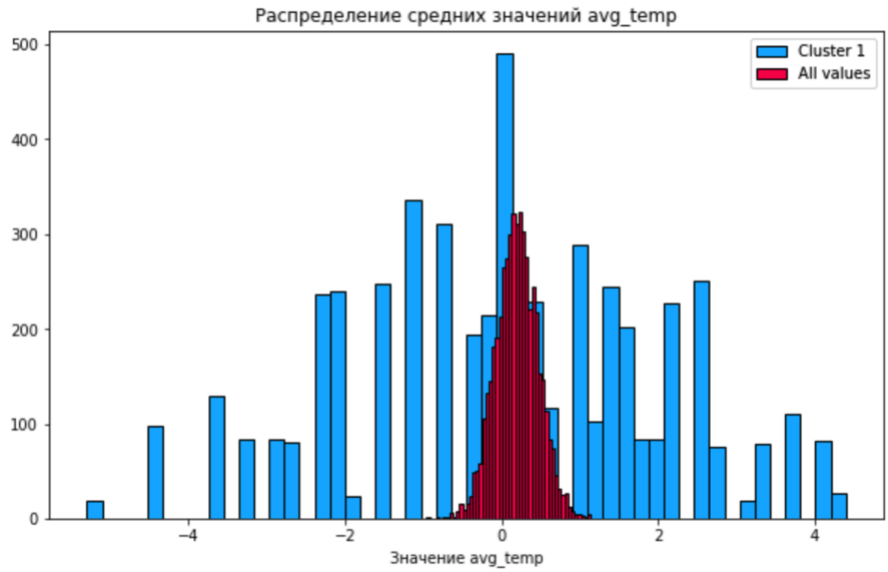

Иллюстрация 9 - Распределение среднего внутрикластерного значения признака avg_temp в кластере №1 и на всей выборке

Методика расчета доверительных интервалов абсолютно аналогична приведенной выше, поэтому ниже будут приведены только конечные результаты:

```
>>>Границы 95% доверительного интервала:
>>>    Левая:  -3.8915531335149858
>>>    Правая: 3.684264305177112
>>>
>>>Границы 95% доверительного интервала:
>>>    Левая:  -3.9516607523396794
>>>    Правая: 3.747105384492268
```

ТАБЛИЦА СОПРЯЖЕННОСТИ

Сформируем три номинальных признака x1, x2 и x3. В качестве первого номинального признака возьмем разбиение диапазона значений признака `avg_temp` на 4 примерно равных по количеству объектов интервала.

```python
# Разбиение
temp_sorted = df['avg_temp'].sort_values()
average_temp_borders =[
    temp_sorted.iloc[0],
    temp_sorted.iloc[len(temp_sorted.index) // 4],
    temp_sorted.iloc[(len(temp_sorted.index) // 4) * 2],
    temp_sorted.iloc[(len(temp_sorted.index) // 4) * 3],
    temp_sorted.iloc[len(temp_sorted.index)-1] + 0.1
]
# Визуализация
fig, graph = plt.subplots(figsize=(10, 6))
plt.hist(df['avg_temp'], bins=15, color='#00b5ff', label='avg_temp', edgecolor='black')
plt.axvline(x=average_temp_borders[1], color='#f91155')
plt.axvline(x=average_temp_borders[2], color='#f91155')
plt.axvline(x=average_temp_borders[3], color='#f91155')
# Вывод
print(f'''''
    Границы интервала: [{average_temp_borders[0]}; {average_temp_borders[1]});\tЗначений в
 интервале: {len(df[(df.avg_temp >= average_temp_borders[0]) & (df.avg_temp < average_temp
_borders[1])])}
```

```
    Границы интервала: [{average_temp_borders[1]}; {average_temp_borders[2]});\tЗначений в интервале: {len(df[(df.avg_temp >= average_temp_borders[1]) & (df.avg_temp < average_temp_borders[2])])}
    Границы интервала: [{average_temp_borders[2]}; {average_temp_borders[3]});\tЗначений в интервале: {len(df[(df.avg_temp >= average_temp_borders[2]) & (df.avg_temp < average_temp_borders[3])])}
    Границы интервала: [{average_temp_borders[3]}; {average_temp_borders[4]});\tЗначений в интервале: {len(df[(df.avg_temp >= average_temp_borders[3]) & (df.avg_temp < average_temp_borders[4])])}
''')
```

```
Границы интервала: [-16.9; -2.9);    Значений в интервале: 93
Границы интервала: [-2.9; 0.3);      Значений в интервале: 89
Границы интервала: [0.3; 3.8);       Значений в интервале: 94
Границы интервала: [3.8; 12.5);      Значений в интервале: 99
```

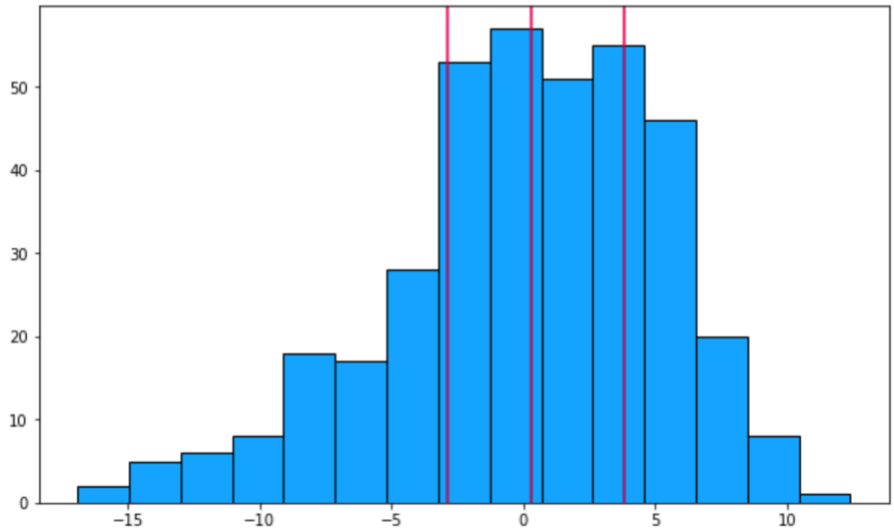

Иллюстрация 10 - Разбиение объектов по признаку avg_temp на 4 приблизительно равных по количеству объектов интервала

Для выделения второго номинального признака рассмотрим распределение признака `snow_depth`:

```
fig, graph = plt.subplots(figsize=(10, 6))
plt.hist(df['snow_depth'], bins=30, color='#00b5ff', label='snow_depth', edgecolor='black')

snow_depth_borders = [0, 0.1, 5, 13, 27]
print(f'''''Рассмотрим распределение, и выберем для него границы интервалов
    Границы интервала: [{snow_depth_borders[0]}; {snow_depth_borders[1]});\tЗначений в интервале: {len(df[(df.snow_depth >= snow_depth_borders[0]) & (df.snow_depth < snow_depth_borders[1])])}
    Границы интервала: [{snow_depth_borders[1]}; {snow_depth_borders[2]});\tЗначений в интервале: {len(df[(df.snow_depth >= snow_depth_borders[1]) & (df.snow_depth < snow_depth_borders[2])])}
    Границы интервала: [{snow_depth_borders[2]}; {snow_depth_borders[3]});\t\tЗначений в интервале: {len(df[(df.snow_depth >= snow_depth_borders[2]) & (df.snow_depth < snow_depth_borders[3])])}
    Границы интервала: [{snow_depth_borders[3]}; {snow_depth_borders[4]});\tЗначений в интервале: {len(df[(df.snow_depth >= snow_depth_borders[3]) & (df.snow_depth < snow_depth_borders[4])])}
''')

plt.axvline(x=snow_depth_borders[1], color='#f91155')
plt.axvline(x=snow_depth_borders[2], color='#f91155')
plt.axvline(x=snow_depth_borders[3], color='#f91155')
```

```
Рассмотрим распределение, и выберем для него границы интервалов
    Границы интервала: [0; 0.1);        Значений в интервале: 249
    Границы интервала: [0.1; 5);        Значений в интервале: 46
    Границы интервала: [5; 13);         Значений в интервале: 56
    Границы интервала: [13; 27);        Значений в интервале: 24
```

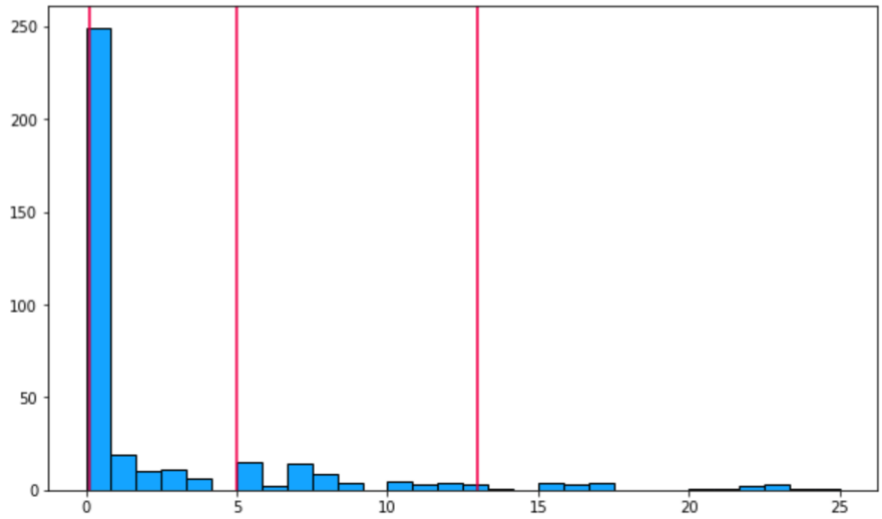

Иллюстрация 11 - Разбиение объектов по признаку snow_depth на 4 интервала

Для выделения третьего номинального признака рассмотрим распределение признака `downfall`:

```
fig, graph = plt.subplots(figsize=(10, 6))
plt.hist(df['downfall'], bins=30, color='#00b5ff', label='downfall', edgecolor='black')

downfall_borders = [0, 0.1, 4, 7, 28]
print(f'''''Рассмотрим распределение, и выберем для него границы интервалов
    Границы интервала: [{downfall_borders[0]}; {downfall_borders[1]});\tЗначений в интерва
ле: {len(df[(df.downfall >= downfall_borders[0]) & (df.downfall < downfall_borders[1])])}
    Границы интервала: [{downfall_borders[1]}; {downfall_borders[2]});\tЗначений в интерва
ле: {len(df[(df.downfall >= downfall_borders[1]) & (df.downfall < downfall_borders[2])])}
    Границы интервала: [{downfall_borders[2]}; {downfall_borders[3]});\t\tЗначений в интер
вале: {len(df[(df.downfall >= downfall_borders[2]) & (df.downfall < downfall_borders[3])])
}
    Границы интервала: [{downfall_borders[3]}; {downfall_borders[4]});\t\tЗначений в интер
вале: {len(df[(df.downfall >= downfall_borders[3]) & (df.downfall < downfall_borders[4])])
}
''')

plt.axvline(x=downfall_borders[1], color='#f91155')
plt.axvline(x=downfall_borders[2], color='#f91155')
plt.axvline(x=downfall_borders[3], color='#f91155')
```

```
Рассмотрим распределение, и выберем для него границы интервалов
    Границы интервала: [0; 0.1);        Значений в интервале: 245
    Границы интервала: [0.1; 4);        Значений в интервале: 91
    Границы интервала: [4; 7);          Значений в интервале: 28
    Границы интервала: [7; 28);         Значений в интервале: 11
```

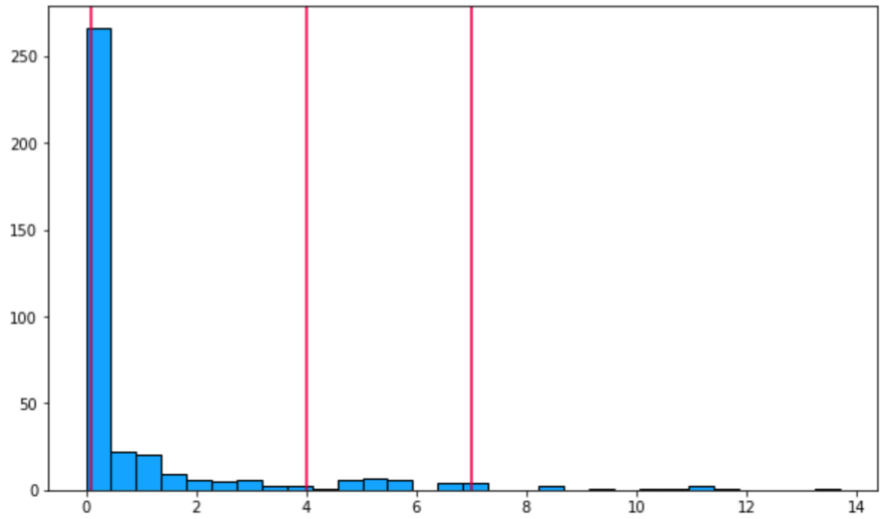

Иллюстрация 12 - Разбиение объектов по признаку downfall на 4 интервала

Сформируем две таблицы сопряженности, x1 и x2, x1 и x3:

```python
# Таблица сопряженности x1(avg_temp) и x2(snow_depth)
t_s = np.arange(16).reshape(4, 4)
# Таблица сопряженности x1(avg_temp) и x3(downfall)
t_d = np.arange(16).reshape(4, 4)

for i in range(4):
    for j in range(4):
        t_s[j][i] = len(
            df[
                (snow_depth_borders[i] <= df['snow_depth']) &
                (df['snow_depth'] < snow_depth_borders[i+1]) &
                (average_temp_borders[j] <= df['avg_temp']) &
                (df['avg_temp'] < average_temp_borders[j+1])
            ].index
        )
        t_d[j][i] = len(
            df[
                (downfall_borders[i] <= df['downfall']) &
                (df['downfall'] < downfall_borders[i+1]) &
                (average_temp_borders[j] <= df['avg_temp']) &
                (df['avg_temp'] < average_temp_borders[j+1])
            ].index
        )
```

		downfall				
		[0, 0.1)	**[0.1, 4)**	**[4, 7)**	**[7, 28)**	**Total**
avg_temp	**[-16.9, -2.2)**	69	20	3	1	93
	[-2.2, 1.0)	54	21	10	4	89
	[1.0, 3.6)	59	25	6	4	94
	[3.6, 14.0)	63	25	9	2	99
	Total	245	91	28	11	375

Таблица 2 - Таблица сопряженности x1(avg_temp) и x3(downfall)

		snow_depth				
		[0, 0.1)	**[0.1, 5)**	**[5, 13)**	**[13, 27)**	**Total**
avg_temp	**[-16.9, -2.2)**	32	12	31	18	93
	[-2.2, 1.0)	45	21	18	5	89
	[1.0, 3.6)	73	13	7	1	94
	[3.6, 14.0)	99	0	0	0	99
	Total	249	46	56	24	375

Таблица 3 - Таблица сопряженности x1(avg_temp) и x2(snow_depth)

Построим таблицы условных вероятностей (x1 от x2 и x1 от x3):

```
conditional_probability_t_s = np.zeros((4, 4))
conditional_probability_t_d = np.zeros((4, 4))

for i in range(4):
    for j in range(4):
      conditional_probability_t_s[j,i] = round(t_s[j,i] / sum(t_s[:,i]), 2)
      conditional_probability_t_d[j,i] = round(t_d[j,i] / sum(t_d[:,i]), 2)
```

		snow_depth			
		[0, 0.1)	**[0.1, 5)**	**[5, 13)**	**[13, 27)**
avg_temp	**[-16.9, -2.2)**	0,1285	0,2609	0,5536	0,7500
	[-2.2, 1.0)	0,1807	0,4565	0,3214	0,2083
	[1.0, 3.6)	0,2932	0,2826	0,1250	0,0417
	[3.6, 14.0)	0,3976	0,0000	0,0000	0,0000

Таблица 4 - Таблица условных вероятностей x1(avg_temp) и x2(snow_depth)

		downfall			
		[0, 0.1)	**[0.1, 4)**	**[4, 7)**	**[7, 28)**
avg_temp	**[-16.9, -2.2)**	0,2816	0,2198	0,1071	0,0909
	[-2.2, 1.0)	0,2204	0,2308	0,3571	0,3636
	[1.0, 3.6)	0,2408	0,2747	0,2143	0,3636
	[3.6, 14.0)	0,2571	0,2747	0,3214	0,1818

Таблица 5 - Таблица условных вероятностей x1(avg_temp) и x3(downfall)

Построим таблицы коэффициентов Кетле:

```
probability_t_s = np.zeros((4, 4))
probability_t_d = np.zeros((4, 4))

for i in range(4):
    for j in range(4):
      probability_t_s[j,i] = round(t_s[j,i] / t_s.sum(), 2)
      probability_t_d[j,i] = round(t_d[j,i] / t_d.sum(), 2)

ketle_t_s = np.zeros((4, 4))
ketle_t_d = np.zeros((4, 4))

for i in range(4):
    for j in range(4):
      ketle_t_s[j,i] = round((probability_t_s[j,i] / ( sum(probability_t_s[:,i]) * sum(pro
bability_t_s[j,:] )) - 1)*100, 2)
      ketle_t_d[j,i] = round((probability_t_d[j,i] / ( sum(probability_t_d[:,i]) * sum(pro
bability_t_d[j,:] )) - 1)*100, 2)
```

		snow_depth			
		[0, 0.1)	**[0.1, 5)**	**[5, 13)**	**[13, 27)**
avg_temp	**[-16.9, -2.2)**	-48,1798	5,1893	123,2143	202,4194
	[-2.2, 1.0)	-23,8527	92,3547	35,4334	-12,2191
	[1.0, 3.6)	16,9572	12,7428	-50,1330	-83,3777
	[3.6, 14.0)	50,6024	-100,0000	-100,0000	-100,0000

Таблица 6 - Таблица коэффициентов Кетле x1(avg_temp) и x2(snow_depth)

		downfall			
		[0, 0.1)	**[0.1, 4)**	**[4, 7)**	**[7, 28)**
avg_temp	**[-16.9, -2.2)**	13,5616	-11,3789	-56,7972	-63,3431
	[-2.2, 1.0)	-7,1314	-2,7658	50,4815	53,2176
	[1.0, 3.6)	-3,9297	9,5978	-14,5137	45,0677
	[3.6, 14.0)	-2,5974	4,0626	21,7532	-31,1295

Таблица 7 - Таблица коэффициентов Кетле x1(avg_temp) и x3(downfall)

Как видно из таблиц, средняя глубина снега растёт с уменьшением средней температуры, а осадки наиболее вероятны при температуре около нуля.

Вычислим и визуализируем среднее значение индекса Кетле на построенных таблицах сопряженности:

```
ind_ketle_t_s = np.zeros((4, 4))
ind_ketle_t_d = np.zeros((4, 4))

for i in range(4):
    for j in range(4):
      ind_ketle_t_s[j,i] = probability_t_s[j,i]*ketle_t_s[j,i]/100
      ind_ketle_t_d[j,i] = probability_t_d[j,i]*ketle_t_s[j,i]/100

print(f'''''
Среднее значение индекса Кетле x1(avg_temp) и x2(snow_depth): {ind_ketle_t_s.sum()}
Среднее значение индекса Кетле x1(avg_temp) и x3(downfall): {ind_ketle_t_d.sum()}
''')
```

		snow_depth				
		[0, 0.1)	**[0.1, 5)**	**[5, 13)**	**[13, 27)**	**Total**
avg_temp	**[-16.9, -2.2)**	-0,0411	0,0017	0,1019	0,0972	0,1596
	[-2.2, 1.0)	-0,0286	0,0517	0,0170	-0,0016	0,0385
	[1.0, 3.6)	0,0330	0,0044	-0,0094	-0,0022	0,0258
	[3.6, 14.0)	0,1336	0,0000	0,0000	0,0000	0,1336
	Total	0,0969	0,0578	0,1095	0,0933	0,3575

Таблица 8 - Таблица индексов Кетле x1(avg_temp) и x2(snow_depth) и среднее значение индекса Кетле (выделено желтым)

		downfall				
		[0, 0.1)	**[0.1, 4)**	**[4, 7)**	**[7, 28)**	**Total**
avg_temp	**[-16.9, -2.2)**	0,0250	-0,0061	-0,0045	-0,0017	0,0127
	[-2.2, 1.0)	-0,0103	-0,0015	0,0135	0,0057	0,0073
	[1.0, 3.6)	-0,0062	0,0064	-0,0023	0,0048	0,0027
	[3.6, 14.0)	-0,0044	0,0027	0,0052	-0,0017	0,0019
	Total	0,0041	0,0015	0,0118	0,0071	0,0246

Таблица 9 - Таблица индексов Кетле x1(avg_temp) и x3(downfall) и среднее значение индекса Кетле (выделено желтым)

Прокомментируем смысл значений индекса Кетле на нескольких примерах:

Для температуры больше 3.5 градусов и глубины снега больше 0 значение индекса Кетле -100, т.е. такого не бывает, что вполне логично, так как снег при такой температуре просто растает. Для низкой температуры и глубокого снега значение индекса Кетле очень высокое, т.е. эти события почти всегда происходят одновременно.

Для большого количества осадков и низкой температуры индекс Кетле очень низкий, а для температуры около 0 и осадков индекс Кетле высокий, т.е. осадки наиболее вероятны при температуре у 0, и крайне маловероятны при морозах.

МГК/SVD

Из наших данных более всего под понятие «более или менее относящиеся к одному и тому же аспекту данных», подходят признаки:

- `avg_temp` – действительное число – средняя температура в течении дня (в градусах Цельсия);
- `min_temp` – действительное число – минимальная температура в течении дня (в градусах Цельсия);
- `max_temp` – действительное число – максимальная температура в течении дня (в градусах Цельсия);

Подготовим данные для дальнейшего использования:

```
1. av_t = df[['avg_temp', 'min_temp', 'max_temp']].to_numpy()
2. amplitude_scoring = pd.DataFrame((av_t - av_t.mean(axis = 0))/(av_t.max(axis = 0) - av_t.m
   in(axis = 0)), columns = ['avg_temp', 'min_temp', 'max_temp'])
3. amplitude_scoring['snow_depth'] = df['snow_depth'].to_numpy()
4. z_scoring = pd.DataFrame((av_t - av_t.mean(axis = 0))/np.sqrt( ((av_t - av_t.mean(axis = 0
   ))*(av_t - av_t.mean(axis = 0))).mean(axis = 0) ), columns = ['avg_temp', 'min_temp', 'max
   _temp'])
5. z_scoring['snow_depth'] = df['snow_depth'].to_numpy()
```

Для визуализации данных с использованием МГК и стандартизации на размах будем использовать модуль PCA[9] библиотеки SKLearn[7]. Визуализируем наши данные с

использованием МГК и стандартизации на размах, при этом цветом отметим наличие/отсутствие снега:

```
from sklearn.decomposition import PCA

pca_a = PCA(n_components = 2)
XPCAreduced = pca_a.fit_transform(amplitude_scoring[['avg_temp', 'min_temp', 'max_temp']].
to_numpy())
a_s = pd.DataFrame(XPCAreduced, columns = ['col1', 'col2'])
a_s['snow_depth'] = df['snow_depth'].to_numpy()

fig, graph = plt.subplots(figsize=(10, 6))
plt.scatter(
        a_s[a_s['snow_depth'] == 0]['col1'],
        a_s[a_s['snow_depth'] == 0]['col2'],
        label='Нет снега',
        cmap='rainbow'
    )
plt.scatter(
        a_s[a_s['snow_depth'] > 0]['col1'],
        a_s[a_s['snow_depth'] > 0]['col2'],
        label='Есть снег',
        cmap='rainbow'
    )
plt.title('Визуализация данных с использованием МГК и стандартизации на размах')
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.grid()
plt.legend(title='Снег')
```

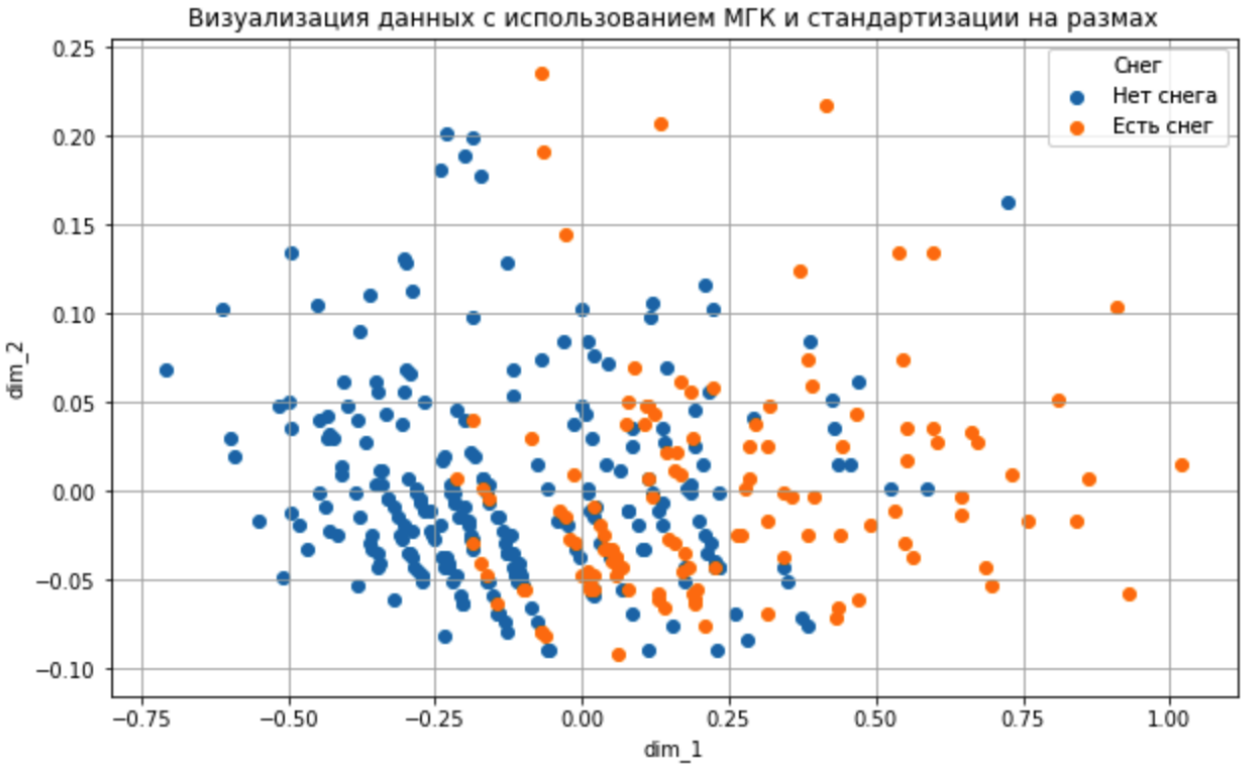

Иллюстрация 13 - Визуализация данных с использованием МГК и стандартизации на размах

Теперь визуализируем наши данные с использованием МГК и z-scoring, при этом цветом отметим наличие/отсутствие снега:

```
from sklearn.decomposition import PCA

pca_z = PCA(n_components = 2)
XPCAreduced = pca_z.fit_transform(z_scoring[['avg_temp', 'min_temp', 'max_temp']].to_numpy
())
z_s = pd.DataFrame(XPCAreduced, columns = ['col1', 'col2'])
z_s['snow_depth'] = df['snow_depth'].to_numpy()

fig, graph = plt.subplots(figsize=(10, 6))
plt.scatter(
        z_s[z_s['snow_depth'] == 0]['col1'],
        z_s[z_s['snow_depth'] == 0]['col2'],
        label='Нет снега',
        cmap='rainbow'
    )
plt.scatter(
        z_s[z_s['snow_depth'] > 0]['col1'],
        z_s[z_s['snow_depth'] > 0]['col2'],
        label='Снег',
        cmap='rainbow'
    )

plt.title('Визуализация данных с использованием МГК и z-scoring')
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.grid()
plt.legend(title='Снег')
```

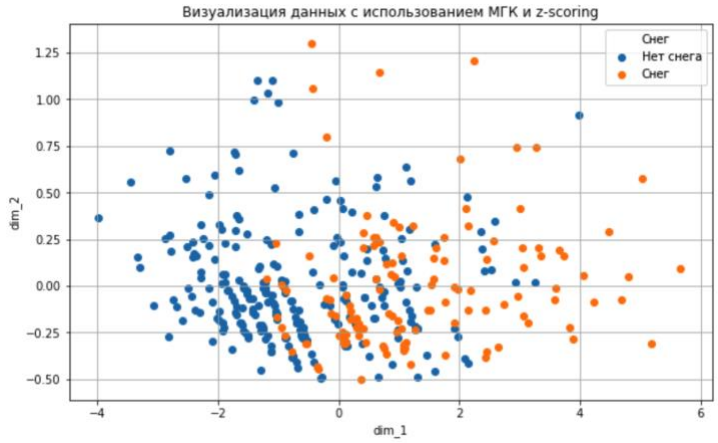

Иллюстрация 14 - Визуализация данных с использованием МГК и z-scoring

А сейчас применим традиционный метод МГК для визуализации этих же данных и сравним с результат с использованными выше библиотечными методами:

```
X = np.transpose(df[['avg_temp', 'min_temp', 'max_temp']].to_numpy())
Xcentered = (X[0] - X[0].mean(), X[1] - X[1].mean(), X[2] - X[2].mean())
m = (X[0].mean(), X[1].mean(), X[2].mean())
covmat = np.cov(Xcentered)
_, vecs = np.linalg.eig(covmat)
v = [vecs[:,0], vecs[:,2]]
Xnew = pd.DataFrame(np.transpose(np.dot(v,Xcentered)), columns = ['col1', 'col2'])
Xnew['snow_depth'] = df['snow_depth'].to_numpy()

fig, graph = plt.subplots(figsize=(10, 6))
plt.scatter(
        Xnew[Xnew['snow_depth'] == 0]['col1'],
        Xnew[Xnew['snow_depth'] == 0]['col2'],
        label='Нет снега',
        cmap='rainbow'
    )
plt.scatter(
        Xnew[Xnew['snow_depth'] > 0]['col1'],
```

```
        Xnew[Xnew['snow_depth'] > 0]['col2'],
        label='Есть снег',
        cmap='rainbow'
    )

plt.title('Визуализация данных с использованием традиционного МГК, реализованного вручную'
)
plt.legend(title='Снег')
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.grid()
```

Иллюстрация 15 - Визуализация данных с использованием традиционного МГК, реализованного вручную

С помощью традиционного МКГ, реализованного вручную, мы получили такой же результат, как и при использовании стандартизации на размах и z-scoring, только отзеркаленный по одной из осей.

Рассмотрим главные компоненты:

```
print ("Первая главная компонента: ", v[0], "\nВторая главная компонента: ", v[1])

>>>Первая главная компонента:  [0.57942185 0.60603756 0.54496678]
>>>Вторая главная компонента:  [ 0.01553545 -0.67673907  0.73605902]
```

В первой главной компоненте коэффициенты почти равны, таким образом первая компонента — это некий общий показатель температуры, на основе всех трёх температурных признаков.

Во второй главной компоненте коэффициент перед среднесуточной температурой практически равен 0, а коэффициенты перед минимальной и максимальной температурой почти равны по модулю и противоположны по знаку, значит это разность минимальной и максимальной температуры (умноженная на какой-то коэффициент), т.е. амплитуда колебания температуры в течении суток.

ЛИНЕЙНАЯ РЕГРЕССИЯ

Попробуем найти в наших данных два признака с более-менее линейным полем рассеяния. Для этого воспользуемся функцией pairplot[10] библиотеки seaborn[11].

```
1. import seaborn as sns
2.
3. sns.set()
4. cols = ['date','avg_temp','min_temp','max_temp','downfall','pressure','humidity','wind_speed','snow_depth','year']
5. sns.pairplot(df[cols])
```

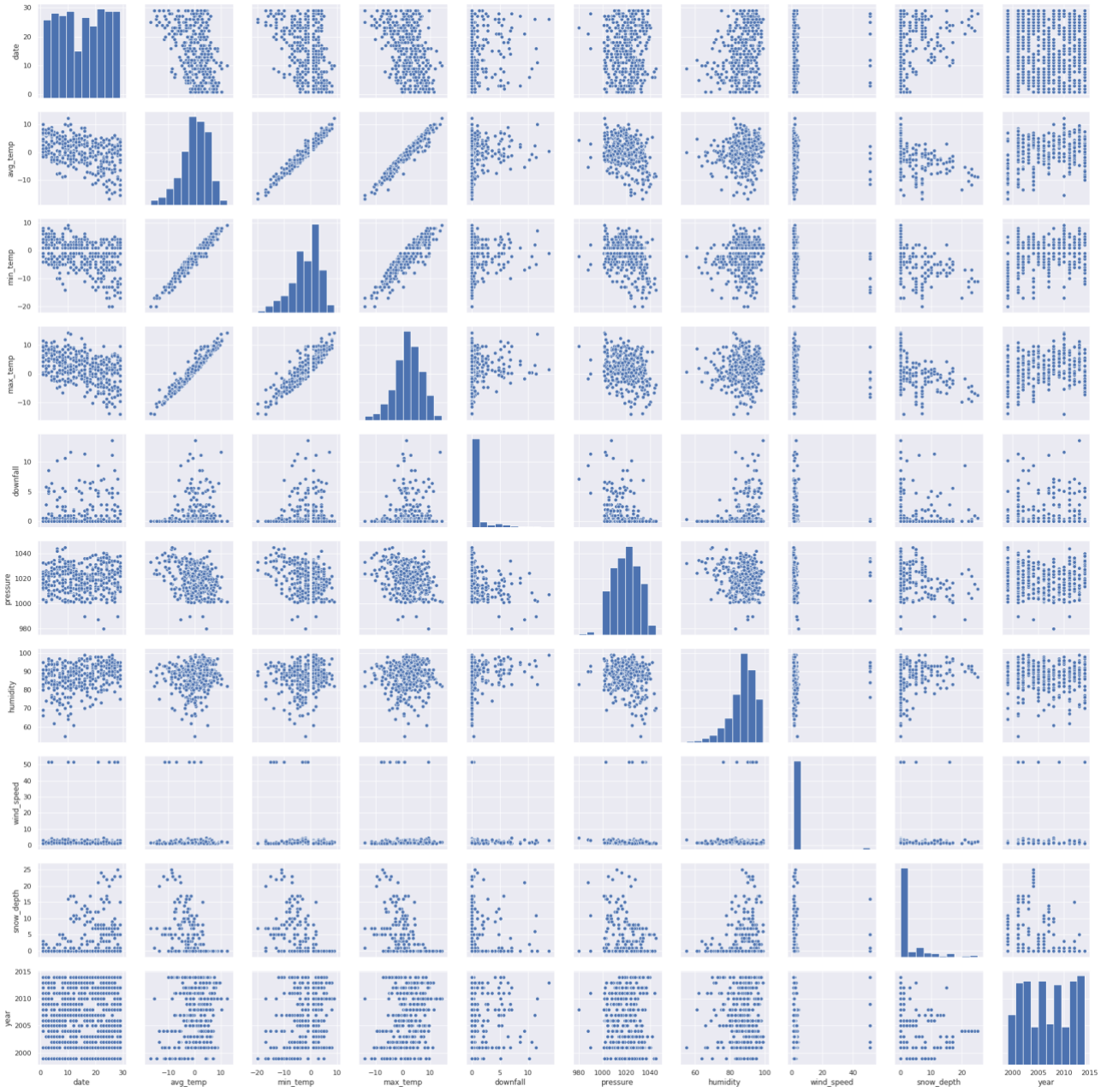

Иллюстрация 16 - Взаимосвязи между всеми признаками в наших данных

Если не считать взаимосвязи между `avg_temp`, `min_temp` и `max_temp` (логично, что они все описывают температуру), то больше остальных на линейное поле рассеяния похожа взаимосвязь между date и avg_temp (логично, ибо чем ближе к зиме – тем холоднее). Далее будем рассматривать именно эту пару признаков.
Рассмотрим это поле рассеивания более подробно:

```
fig, graph = plt.subplots(figsize=(13, 9))
for i in range(1, 31):
  plt.scatter(
        df[df.date == i].date,
        df[df.date == i].avg_temp,
        label=f'{i} ноября',
        cmap='rainbow'
    )
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

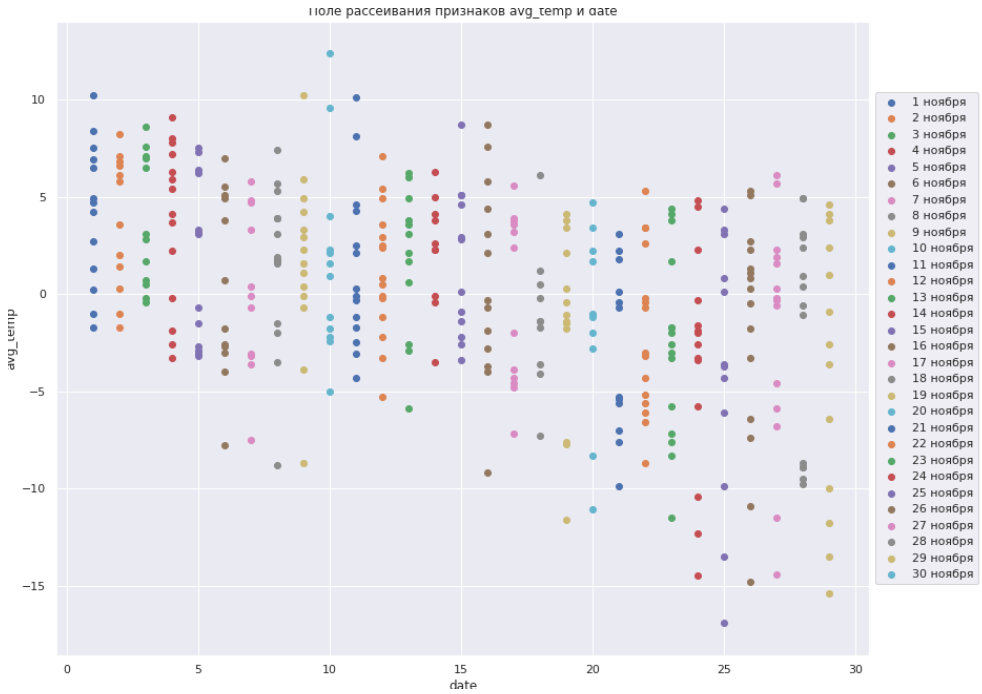

Иллюстрация 17 - Поле рассеивания признаков avg_temp и date

Обучим линейную регрессию:

- date - признак
- avg_temp - целевое значение

Для реализации линейной регрессии будем использовать проверенный временем модуль LinearRegression[12] библиотеки SKLearn[7].

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(np.array(df['date'])[:,np.newaxis], np.array(df['avg_temp'])[:,np.newaxis])

print(f'Уравнение линейной регрессии имеет вид:\ty = {model.coef_.item()} * x + {model.intercept_.item()}')

>>>Уравнение линейной регрессии имеет вид: y = -0.2550634686459896 * x + 3.9509900360879273
```

Визуализируем результат:

```
fig, graph = plt.subplots(figsize=(13, 9))
for i in range(1, 31):
  plt.scatter(
        df[df.date == i].date,
        df[df.date == i].avg_temp,
```

```
        label=f'{i} ноября',
        cmap='rainbow'
    )

x = np.linspace(0,30)
y = model.coef_.item()*x + model.intercept_.item()
plt.plot(x, y, label='Уравнение линейной регрессии')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

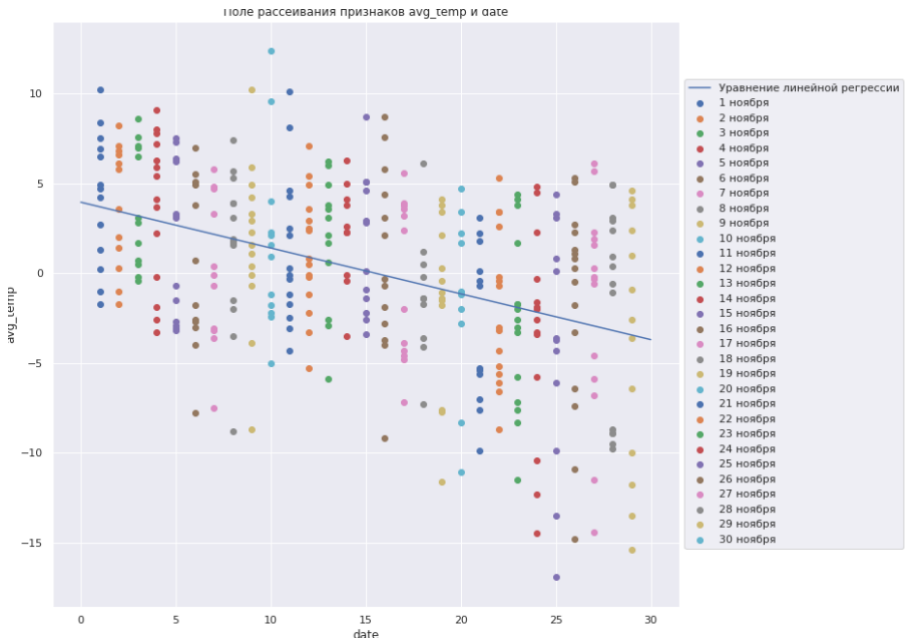

Иллюстрация 18 - Поле рассеивания признаков avg_temp и date, уравнение линейной регрессии

Найдем значения коэффициентов корреляции и детерминации. Для первого воспользуемся функцией pearsonr[13] библиотеки SKLearn[7], а для коэффициента детерминации (также известен в машинном обучении как R^2) – функцией r2_score[14] библиотеки SKLearn[7].

```
from scipy.stats import pearsonr
from sklearn.metrics import r2_score

corr = pearsonr(model.coef_.item()*df['date'] + model.intercept_.item(), df['avg_temp'])[0
]
r2 = r2_score(df['avg_temp'], model.coef_.item()*df['date'] + model.intercept_.item())

print(f'Значение коэффициента корреляции Пирсона:\t{corr}')
print(f'Значение коэффициента детерминации:\t{r2}')

>>>Значение коэффициента корреляции Пирсона:   0.4204443991275562
>>>Значение коэффициента детерминации:     0.1767734927577317
```

Известно, что чем значение коэффициента детерминации к единице, тем точнее предсказание. В нашем же случае он ближе к нулю, чем к единице, и значит предсказание не самое точное. Сделаем интересное наблюдение, а именно - проверим, какую температуру предскажет модель для нескольких чисел ноября, и сравним её с действительностью за окном.

```
print(f'''''
1 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): -
0.5`\tПредсказание:\t{model.predict(np.array([1])[:,np.newaxis]).item()}`
```

```
3.  8 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): +3˚\tПредсказание:\t{model.
    predict(np.array([8])[:,np.newaxis]).item()}˚
4.  15 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): +3.5˚\tПредсказание:\t{mod
    el.predict(np.array([15])[:,np.newaxis]).item()}˚
5.  22 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): -
    3.5˚\tПредсказание:\t{model.predict(np.array([22])[:,np.newaxis]).item()}
6.  ''')
7.
8.  1 ноября 2019   Средняя дневная температура (по Яндекс.Погода): -
    0.5˚   Предсказание:   3.695926567441938˚
9.  8 ноября 2019   Средняя дневная температура (по Яндекс.Погода): +3˚ Предсказание:   1.9104
    822869200104˚
10. 15 ноября 2019  Средняя дневная температура (по Яндекс.Погода): +3.5˚   Предсказание:   0.
    12503800639808293˚
11. 22 ноября 2019  Средняя дневная температура (по Яндекс.Погода): -3.5˚   Предсказание:   -
    1.6604062741238446˚
```

Предсказание не то, чтобы очень точное, но это и очень грубое приближение в конце концов. А по данным модели, в целом, тренд прослеживается, и примерно понятно, когда пора надеть свитер.

Рассчитаем среднюю относительную ошибку регрессионного уравнения на всех объектах нашей таблицы данных:

```
1.  tmp = df[['date', 'avg_temp']]
2.  tmp['avg_temp_predict'] = tmp.apply(lambda x: x.date * model.coef_.item() + model.intercep
    t_.item(), axis=1)
3.  tmp['diff_percentage'] = tmp.apply(lambda x: abs((x.avg_temp - x.avg_temp_predict) * 100.0
     / x.avg_temp), axis=1)
4.  print(f'Значение средней относительной ошибки: {tmp.diff_percentage.mean()}%')
5.
6.  >>>Значение средней относительной ошибки: 175.4859768883784%
```

Собственно, как и было видно ранее из значения коэффициента детерминации, линейная регрессия от одного признака дает очень грубое приближение. Мало того, что погода – вещь крайне непредсказуемая (особенно в Москве), так мы еще и пытаемся приблизить её линейной функцией от одного аргумента.

СПИСОК ИСПОЛЬЗОВАННЫХ МАТЕРИАЛОВ

[1] Источник датасета с погодой в Москве - https://climate-energy.ru/weather/archive_weather_276120.php

[2] Датасет, использованный в ходе работы https://github.com/mgcrp/hse_ccda_2019/blob/master/CLEAR_dataset_november_weather.csv

[3] Среда для разработки Google Colab - https://colab.research.google.com

[4] Pandas – Python-библиотека для анализа данных - https://pandas.pydata.org

[5] NumPy – Python-библиотека для матричных и векторных операций https://numpy.org

[6] MatPlotLib – Python-библиотека для построения графиков и визуализации - https://matplotlib.org

[7] SKLearn – Python-библиотека для машинного обучения и анализа данных https://scikit-learn.org/stable/

[8] Модуль Kmeans библиотеки SKLearn – https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[9] Модуль PCA библиотеки SKLearn – https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[10] https://seaborn.pydata.org/generated/seaborn.pairplot.html

[11] Seaborn – Python-библиотека для продвинутого построения графиков и визуализации - https://seaborn.pydata.org/index.html

[12] Модуль LinearRegression библиотеки SKLearn - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[13] Функция pearsonr библиотеки SKLearn - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html

[14] Функция r2_score библиотеки SKLearn - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

ПРИЛОЖЕНИЯ

Приложение 1 – Репозиторий GitHub с материалами

https://github.com/mgcrp/hse_ccda_2019

Приложение 2 – Jupyter Notebook с исходным кодом

https://github.com/mgcrp/hse_ccda_2019/blob/master/homework_final.ipynb

Приложение 3 – Полный исходный код домашней работы

```
# -*- coding: utf-8 -*-
"""HW - Миркин.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/18owHBnh0OKAUPuZUXtP5w5tWwHhsYxXb

# Курсовой проект - Миркин

__Группа__ - Никифоров, Попов
"""

# Подключаю свой Google Drive, там лежит датасет
from google.colab import drive
drive.mount('/content/gdrive')

# Импортирую датасет
import pandas as pd
df = pd.read_csv('gdrive/My Drive/Colab Notebooks/november_weather.csv', delimiter='\t')

# Загрузить датасет
import io
from google.colab import files

uploaded = files.upload()

df = pd.read_csv(io.BytesIO(uploaded['november_weather.csv']), delimiter='\t')

"""# Задание 2, Кластер-анализ
1. Выберите не менее трех количественных признаков, объясните выбор и примените метод K-средних:
  * для K=5
  * для K=9
  * в обоих случаях сделайте порядка 10 случайных инициализаций и выберите то, которое дос тавляет минимум критерию метода
2. Проинтерпретируйте оба разбиения с помощью признаков таблицы данных путем сравнения вну трикластерных средних с общими средними. Объясните, какое из разбиений лучше с точки зрени я интерпретации.

---

__Выбираем признаки__:
 * `avg_temp` - средняя температура в течении дня;
 * `pressure` - среднее атмосферное давление в течении дня;
 * `wind_speed` - срkдняя скорость ветра;
"""

```

```
df_features = df[['avg_temp', 'pressure', 'wind_speed']].drop(df[df.pressure == '-
'].index).drop(df[df.wind_speed > 45].index)
df_features.pressure = df_features.apply(lambda x: float(x.pressure), axis=1)

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
from random import choices
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from statistics import mean, stdev

# %matplotlib inline

"""__1.1 - kmeans для 5 кластеров__"""

kmeans_5 = KMeans(
    n_clusters = 5,        # 5 кластеров
    init = 'random',       # центры кластеров - случайно выбранные точки
    n_init = 10            # количество инициализаций алгоритма
)
kmeans_5.fit(df_features) # выполнение алгоритма

# Добавлю к исходному датасету созданные кластеры
df_features_5 = df_features
df_features['cluster'] = kmeans_5.labels_

# Визуализация
fig, graphs = plt.subplots(1, 3, figsize=(17.5, 5))

plt.axes(graphs[0])
for i in range(5):
    plt.scatter(
        df_features_5[df_features_5.cluster == i].avg_temp,
        df_features_5[df_features_5.cluster == i].pressure,
        label=i+1,
        cmap='rainbow'
    )
plt.xlabel('avg_temp')
plt.ylabel('pressure')
plt.legend(title='Cluster')

plt.axes(graphs[1])
for i in range(5):
    plt.scatter(
        df_features_5[df_features_5.cluster == i].avg_temp,
        df_features_5[df_features_5.cluster == i].wind_speed,
        label=i+1,
        cmap='rainbow'
    )
plt.xlabel('avg_temp')
plt.ylabel('wind_speed')
plt.legend(title='Cluster')

plt.axes(graphs[2])
for i in range(5):
    plt.scatter(
                df_features_5[df_features_5.cluster == i].wind_speed,
                df_features_5[df_features_5.cluster == i].pressure,
                label=i+1,
                cmap='rainbow'
            )
      plt.xlabel('wind_speed')
      plt.ylabel('pressure')
      plt.legend(title='Cluster')

      fig.savefig('gdrive/My Drive/Colab Notebooks/task2_five_clusters.png')
      print()

      """__1.2 - kmeans для 9 кластеров__"""
```

```
113.
114.        kmeans_9 = KMeans(
115.            n_clusters = 9,       # 9 кластеров
116.            init = 'random',      # центры кластеров - случайно выбранные точки
117.            n_init = 10           # количество инициализаций алгоритма
118.        )
119.        kmeans_9.fit(df_features) # выполнение алгоритма
120.
121.        # Добавлю к исходному датасету созданные кластеры
122.        df_features_9 = df_features
123.        df_features['cluster'] = kmeans_9.labels_
124.
125.        # Визуализация
126.        fig, graphs = plt.subplots(1, 3, figsize=(17.5, 5))
127.
128.        plt.axes(graphs[0])
129.        for i in range(9):
130.            plt.scatter(
131.                df_features_9[df_features_9.cluster == i].avg_temp,
132.                df_features_9[df_features_9.cluster == i].pressure,
133.                label=i+1,
134.                cmap='rainbow'
135.            )
136.        plt.xlabel('avg_temp')
137.        plt.ylabel('pressure')
138.        plt.legend(title='Cluster')
139.
140.        plt.axes(graphs[1])
141.        for i in range(9):
142.            plt.scatter(
143.                df_features_9[df_features_9.cluster == i].avg_temp,
144.                df_features_9[df_features_9.cluster == i].wind_speed,
145.                label=i+1,
146.                cmap='rainbow'
147.            )
148.        plt.xlabel('avg_temp')
149.        plt.ylabel('wind_speed')
150.        plt.legend(title='Cluster')
151.
152.        plt.axes(graphs[2])
153.        for i in range(9):
154.            plt.scatter(
155.                df_features_9[df_features_9.cluster == i].wind_speed,
156.                df_features_9[df_features_9.cluster == i].pressure,
157.                label=i+1,
158.                cmap='rainbow'
159.            )
160.        plt.xlabel('wind_speed')
161.        plt.ylabel('pressure')
162.        plt.legend(title='Cluster')
163.
164.        fig.savefig('gdrive/My Drive/Colab Notebooks/task2_nine_clusters.png')
165.        print()
166.
167.        """__2 - Проинтерпретируйте оба разбиения с помощью признаков таблицы данных путем
    сравнения внутрикластерных средних с общими средними. Объясните, какое из разбиений лучше
    с точки зрения интерпретации__"""
168.
169.        metrics_5cluster = pd.DataFrame(columns=['cluster', 'avg_temperature', 'avg_pressur
    e', 'avg_wind'])
170.        metrics_5cluster = metrics_5cluster.append(
171.            pd.Series(
172.                {
173.                    'cluster': 'All data',
174.                    'avg_temperature': mean(df_features_5.avg_temp),
175.                    'avg_pressure': mean(df_features_5.pressure),
176.                    'avg_wind': mean(df_features_5.wind_speed)
177.                }), ignore_index=True)
178.
```

```
    for i in range(5):
        metrics_5cluster = metrics_5cluster.append(
        pd.Series(
            {
                'cluster': f'Cluster #{i}',
                'avg_temperature': mean(df_features_5[df_features_5.cluster == i].avg_temp),
                'avg_pressure': mean(df_features_5[df_features_5.cluster == i].pressure),
                'avg_wind': mean(df_features_5[df_features_5.cluster == i].wind_speed)

            }), ignore_index=True)

    metrics_5cluster = metrics_5cluster.set_index('cluster')
    metrics_5cluster

    # data to plot
    n_groups = 5

    # create plot
    fig, ax = plt.subplots(figsize=(10, 6))
    index = np.arange(n_groups)
    bar_width = 0.2
    opacity = 0.8

    metrics_5cluster_viz = metrics_5cluster[metrics_5cluster.index != 'All data']

    plt.bar(index, metrics_5cluster_viz.avg_temperature - [metrics_5cluster.avg_temperature['All data']]*5, bar_width, label='avg_temperature')
    plt.bar(index + bar_width, metrics_5cluster_viz.avg_pressure - [metrics_5cluster.avg_pressure['All data']]*5, bar_width, label='avg_pressure')
    plt.bar(index + 2*bar_width, metrics_5cluster_viz.avg_wind - [metrics_5cluster.avg_wind['All data']]*5, bar_width, label='avg_wind')

    plt.xlabel('Кластеры')
    plt.ylabel('Разница с общим средним')
    plt.xticks(index + bar_width, metrics_5cluster_viz.index)
    plt.legend()
    plt.grid()

    plt.tight_layout()
    plt.show()

    metrics_9cluster = pd.DataFrame(columns=['cluster', 'avg_temperature', 'avg_pressure', 'avg_wind'])
    metrics_9cluster = metrics_9cluster.append(
        pd.Series(
            {
                'cluster': 'All data',
                'avg_temperature': mean(df_features_9.avg_temp),
                'avg_pressure': mean(df_features_9.pressure),
                'avg_wind': mean(df_features_9.wind_speed)
            }), ignore_index=True)

    for i in range(9):
        metrics_9cluster = metrics_9cluster.append(
        pd.Series(
            {
                'cluster': f'Cluster #{i}',
                'avg_temperature': mean(df_features_9[df_features_9.cluster == i].avg_temp),
                'avg_pressure': mean(df_features_9[df_features_9.cluster == i].pressure),
                'avg_wind': mean(df_features_9[df_features_9.cluster == i].wind_speed)

            }), ignore_index=True)

    metrics_9cluster = metrics_9cluster.set_index('cluster')
    metrics_9cluster
```

```
238.
239.       # data to plot
240.       n_groups = 9
241.
242.       # create plot
243.       fig, ax = plt.subplots(figsize=(10, 6))
244.       index = np.arange(n_groups)
245.       bar_width = 0.2
246.       opacity = 0.8
247.
248.       metrics_9cluster_viz = metrics_9cluster[metrics_9cluster.index != 'All data']
249.
250.       plt.bar(index, metrics_9cluster_viz.avg_temperature - [metrics_9cluster.avg_tempera
    ture['All data']]*9, bar_width, label='avg_temperature')
251.       plt.bar(index + bar_width, metrics_9cluster_viz.avg_pressure - [metrics_9cluster.av
    g_pressure['All data']]*9, bar_width, label='avg_pressure')
252.       plt.bar(index + 2*bar_width, metrics_9cluster_viz.avg_wind - [metrics_9cluster.avg_
    wind['All data']]*9, bar_width, label='avg_wind')
253.
254.       plt.xlabel('Кластеры')
255.       plt.ylabel('Разница с общим средним')
256.       plt.xticks(index + bar_width, metrics_9cluster_viz.index)
257.       plt.legend()
258.       plt.grid()
259.
260.       plt.tight_layout()
261.       plt.show()
262.
263.       """__Вывод__: *yet to do*
264.
265.       # Задание 3 - Бутстреп
266.
267.       Для одного из полученных в д.з. 2 разбиений
268.        * Найдите 95% доверительный интервал для среднего значения какого-
    либо признака на всем множестве объектов, используя бутстрэп
269.        * Сравните средние по какому-либо признаку в двух кластерах, используя бутстрэп
270.        * Для одного из кластеров сравните среднее на всем множестве для какого-
    либо признака с его средним внутри кластера, используя бутстрэп
271.
272.       **Примечание**: каждое применение бутстрэпа должно быть обоими методами, с опорой и
     без
273.
274.       __1 - Найдите 95% доверительный интервал для среднего значения какого-
    либо признака на всем множестве объектов, используя бутстрэп__
275.
276.       Анализировать будем признак `avg_temp`
277.       """
278.
279.       from random import choices
280.       from statistics import mean
281.
282.       # Список средних
283.       means = []
284.
285.       # Количество экспериментов
286.       bootstrap_iters = 5000
287.       # Количество выбираемых значений для эксперимента
288.       bootstrap_choices = len(df_features)
289.
290.       for i in range(bootstrap_iters):
291.           # Добавляем в список
292.           means.append(
293.               # Среднее значение
294.               mean(
295.                   choices(
296.                       # Выбираем случайный элемент из значений признака

297.                       df_features_5['avg_temp'].tolist(),
298.                       # k-раз
```

```
                    k=bootstrap_choices
                )
            )
        )

    # Визуализация
    fig, graph = plt.subplots(figsize=(10, 6))
    plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')
    plt.xlabel('Значение avg_temp')
    plt.title('Распределение средних значений avg_temp')

    print()

    """Найдем 95% доверительный интервал способом "без опоры"""

    # Сортируем список со средними
    means_ordered = sorted(means)
    # Отсекаем 2.5% значений снизу и берем наименьший оставшийся
    left = means_ordered[int(len(means_ordered)*0.05/2 - 1)]
    # Отсекаем 2.5% значений сверху и берем наибольший оставшийся
    right = means_ordered[int(len(means_ordered) - (len(means_ordered)*0.05/2) - 1)]

    print(f'''''
      Границы 95% доверительного интервала:
      \tЛевая:\t{left}
      \tПравая:\t{right}
    ''')

    """Найдем 95% доверительный интервал способом "с опорой"""

    from statistics import mean, stdev

    print(f'''''
      Границы 95% доверительного интервала:
      \tЛевая:\t{mean(means) - 1.96*stdev(means)}
      \tПравая:\t{mean(means) + 1.96*stdev(means)}
    ''')

    """__2 - Сравните средние по какому-
либо признаку в двух кластерах, используя бутстрэп__

    Будем использовать полученное ранее разбиение на 5 кластеров

    Пусть сравниваемыми будут кластеры `3` и `4`
    """

    means = []
    bootstrap_iters = 5000

    for i in range(bootstrap_iters):
        means.append(
            # Из среднего значения признака в эксперименте на 3 кластере
            mean(
                choices(
                    df_features_5[df_features_5.cluster == 3]['avg_temp'].tolist(),
                    k=len(df_features_5[df_features_5.cluster == 3])
                    )
                )
            # вычитаем
            -
            # среднее значение признака в эксперименте на 4 кластере
            mean(
                choices(
                    df_features_5[df_features_5.cluster == 4]['avg_temp'].tolist(),
                    k=len(df_features_5[df_features_5.cluster == 4])
                    )
                )
        )

```

```
        # Визуализация
        fig, graph = plt.subplots(figsize=(10, 6))
        plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')
        plt.xlabel('Значение avg_temp')
        plt.title('Распределение средних значений avg_temp')

        print()

        """0 не попадает в это распределение - средние значения признака avg_temp в кластер
    ах `3` и `4` сильно различаются.

        Рассмотрим распределения по-
    отдельности и расчитаем 95% доверительный интервал для полученного выше распеделения.
        """

        means_cluster3 = []
        means_cluster4 = []

        bootstrap_iters = 5000

        for i in range(bootstrap_iters):
            means_cluster3.append(
                mean(
                    choices(
                        df_features_5[df_features_5.cluster == 3]['avg_temp'].tolist(),
                        k=len(df_features_5[df_features_5.cluster == 3])
                        )
                    )
            )
            means_cluster4.append(
                mean(
                    choices(
                        df_features_5[df_features_5.cluster == 4]['avg_temp'].tolist(),
                        k=len(df_features_5[df_features_5.cluster == 4])
                        )
                    )
            )

        fig, graph = plt.subplots(figsize=(10, 6))
        plt.hist(means_cluster3, bins=50, color='#00b5ff', label='Cluster 3', edgecolor='bl
    ack')
        plt.hist(means_cluster4, bins=50, color='#f91155', label='Cluster 4', edgecolor='bl
    ack')
        plt.legend(title='Кластеры')
        plt.xlabel('Значение avg_temp')
        plt.title('Распределение средних значений avg_temp')

        print()

        """Найдем 95% доверительный интервал способом "без опоры""""

        means_ordered = sorted(means)                                                  # С
    ортируем массив со средними
        left = means_ordered[int(len(means_ordered)*0.05/2 - 1)]                       # О
    тсекаем 2.5% значений снизу и берем наименьший оставшийся
        right = means_ordered[int(len(means_ordered) - (len(means_ordered)*0.05/2) - 1)]# О
    тсекаем 2.5% значений сверху и берем наибольший оставшийся
        print(f'''''
          Границы 95% доверительного интервала:
          \tЛевая:\t{left}
          \tПравая:\t{right}
        ''')

        """Найдем 95% доверительный интервал способом "с опорой""""

        print(f'''''
          Границы 95% доверительного интервала:
          \tЛевая:\t{mean(means) - 1.96*stdev(means)}
          \tПравая:\t{mean(means) + 1.96*stdev(means)}
```

```
429.       ''')
430.
431.       """__3 - Для одного из кластеров сравните среднее на всем множестве для какого-
   либо признака с его средним внутри кластера, используя бутстрэп__
432.
433.       Будем использовать полученное ранее разбиение на 5 кластеров
434.
435.       Сравнивать с выборкой будем значения признака на кластере `1`
436.       """
437.
438.       means = []
439.
440.       bootstrap_iters = 5000
441.
442.       for i in range(bootstrap_iters):
443.           means.append(
444.               mean(
445.                   choices(
446.                       df_features_5[df_features_5.cluster == 1]['avg_temp'].tolist(),
447.                       k=len(df_features_5[df_features_5.cluster == 1])
448.                       )
449.                   )
450.               -
451.               mean(
452.                   choices(
453.                       df_features_5['avg_temp'].tolist(),
454.                       k=len(df_features_5)
455.                       )
456.                   )
457.           )
458.
459.       fig, graph = plt.subplots(figsize=(10, 6))
460.       plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')
461.       plt.xlabel('Значение avg_temp')
462.       plt.title('Распределение средних значений avg_temp')
463.
464.       print()
465.
466.       """0 попадает в это распределение - средние значения признака в кластере `1` и на в
   сей выборке похожи.
467.
468.       Рассмотрим распределения по-
   отдельности и расчитаем 95% доверительный интервал для полученного выше распеделения.
469.       """
470.
471.       means_cluster1 = []                                                          # П
   ринцип аналогичен используемому выше
472.       means_allvalues = []
473.
474.       bootstrap_iters = 5000
475.
476.       for i in range(bootstrap_iters):
477.           means_cluster1.append(
478.               mean(
479.                   choices(
480.                       df_features_5[df_features_5.cluster == 1]['avg_temp'].tolist(),
481.                       k=len(df_features_5[df_features_5.cluster == 1])
482.                       )
483.                   )
484.           )
485.           means_allvalues.append(
486.               mean(
487.                   choices(
488.                       df_features_5['avg_temp'].tolist(),
489.                       k=len(df_features_5)
490.                       )
491.                   )
492.           )
493.
```

```
        fig, graph = plt.subplots(figsize=(10, 6))
        plt.hist(means_cluster1, bins=50, color='#00b5ff', label='Cluster 1', edgecolor='black')
        plt.hist(means_allvalues, bins=50, color='#f91155', label='All values', edgecolor='black')
        plt.legend()
        plt.xlabel('Значение avg_temp')
        plt.title('Распределение средних значений avg_temp')

        print()

        """Найдем 95% доверительный интервал способом "без опоры""""

        means_ordered = sorted(means)                                                    # Сортируем массив со средними
        left = means_ordered[int(len(means_ordered)*0.05/2 - 1)]                         # Отсекаем 2.5% значений снизу и берем наименьший оставшийся
        right = means_ordered[int(len(means_ordered) - (len(means_ordered)*0.05/2) - 1)]# Отсекаем 2.5% значений сверху и берем наибольший оставшийся
        print(f'''''
          Границы 95% доверительного интервала:
          \tЛевая:\t{left}
          \tПравая:\t{right}
        ''')

        """Найдем 95% доверительный интервал способом "с опорой""""

        print(f'''''
          Границы 95% доверительного интервала:
          \tЛевая:\t{mean(means) - 1.96*stdev(means)}
          \tПравая:\t{mean(means) + 1.96*stdev(means)}
        ''')

        """# Задание 4 - Таблица сопряженности

        1. Сформируйте три номинальных признака x1, x2 и x3 (один из них, но не больше, может быть взят из Вашей таблицы данных)
        2. Сформируйте две таблицы сопряженности, x1 и x2, x1 и x3. Постройте также матрицы условных вероятностей (x1 от x2 и x1 от x3), а также таблицы коэффициентов Кетле. Прокомментируйте связи между категориями x1 и категориями x2 и x3.
        3. Вычислите и визуализируйте среднее значение индекса Кетле на построенных таблицах сопряженности.
        4. Прокомментируйте смысл значений индекса Кетле на двух-трех примерах.

        __1 - Сформируйте три номинальных признака x1, x2 и x3 (один из них, но не больше, может быть взят из Вашей таблицы данных)__

        В качестве первого номинального признака возьмем разбиение диапазона значений признака `avg_temp` на 4 равных по количеству объектов интревала
        """

        # Разбиение
        temp_sorted = df['avg_temp'].sort_values()
        average_temp_borders =[
            temp_sorted.iloc[0],
            temp_sorted.iloc[len(temp_sorted.index) // 4],
            temp_sorted.iloc[(len(temp_sorted.index) // 4) * 2],
            temp_sorted.iloc[(len(temp_sorted.index) // 4) * 3],
            temp_sorted.iloc[len(temp_sorted.index)-1] + 0.1
        ]
        # Визуализация
        fig, graph = plt.subplots(figsize=(10, 6))
        plt.hist(df['avg_temp'], bins=15, color='#00b5ff', label='avg_temp', edgecolor='black')
        plt.axvline(x=average_temp_borders[1], color='#f91155')
        plt.axvline(x=average_temp_borders[2], color='#f91155')
        plt.axvline(x=average_temp_borders[3], color='#f91155')
        # Вывод
        print(f'''''
```

```
            Границы интервала: [{average_temp_borders[0]}; {average_temp_borders[1]});\t3на
    чений в интервале: {len(df[(df.avg_temp >= average_temp_borders[0]) & (df.avg_temp < avera
    ge_temp_borders[1])])}
            Границы интервала: [{average_temp_borders[1]}; {average_temp_borders[2]});\t3на
    чений в интервале: {len(df[(df.avg_temp >= average_temp_borders[1]) & (df.avg_temp < avera
    ge_temp_borders[2])])}
            Границы интервала: [{average_temp_borders[2]}; {average_temp_borders[3]});\t3на
    чений в интервале: {len(df[(df.avg_temp >= average_temp_borders[2]) & (df.avg_temp < avera
    ge_temp_borders[3])])}
            Границы интервала: [{average_temp_borders[3]}; {average_temp_borders[4]});\t3на
    чений в интервале: {len(df[(df.avg_temp >= average_temp_borders[3]) & (df.avg_temp < avera
    ge_temp_borders[4])])}
        ''')

        """Для выделения второго номинального признака рассмотрим распределение признака `s
    now_depth`"""

        fig, graph = plt.subplots(figsize=(10, 6))
        plt.hist(df['snow_depth'], bins=30, color='#00b5ff', label='snow_depth', edgecolor=
    'black')

        snow_depth_borders = [0, 0.1, 5, 13, 27]
        print(f'''''Рассмотрим распределение, и выберем для него границы интервалов
            Границы интервала: [{snow_depth_borders[0]}; {snow_depth_borders[1]});\t3значени
    й в интервале: {len(df[(df.snow_depth >= snow_depth_borders[0]) & (df.snow_depth < snow_de
    pth_borders[1])])}
            Границы интервала: [{snow_depth_borders[1]}; {snow_depth_borders[2]});\t3значени
    й в интервале: {len(df[(df.snow_depth >= snow_depth_borders[1]) & (df.snow_depth < snow_de
    pth_borders[2])])}
            Границы интервала: [{snow_depth_borders[2]}; {snow_depth_borders[3]});\t\t3значе
    ний в интервале: {len(df[(df.snow_depth >= snow_depth_borders[2]) & (df.snow_depth < snow_
    depth_borders[3])])}
            Границы интервала: [{snow_depth_borders[3]}; {snow_depth_borders[4]});\t3значени
    й в интервале: {len(df[(df.snow_depth >= snow_depth_borders[3]) & (df.snow_depth < snow_de
    pth_borders[4])])}
        ''')

        plt.axvline(x=snow_depth_borders[1], color='#f91155')
        plt.axvline(x=snow_depth_borders[2], color='#f91155')
        plt.axvline(x=snow_depth_borders[3], color='#f91155')

        print()

        """Для выделения третьего номинального признака рассмотрим распределение признака `
    downfall`"""

        fig, graph = plt.subplots(figsize=(10, 6))
        plt.hist(df['downfall'], bins=30, color='#00b5ff', label='downfall', edgecolor='bla
    ck')

        downfall_borders = [0, 0.1, 4, 7, 28]
        print(f'''''Рассмотрим распределение, и выберем для него границы интервалов
            Границы интервала: [{downfall_borders[0]}; {downfall_borders[1]});\t3начений в
    интервале: {len(df[(df.downfall >= downfall_borders[0]) & (df.downfall < downfall_borders[
    1])])}
            Границы интервала: [{downfall_borders[1]}; {downfall_borders[2]});\t3начений в
    интервале: {len(df[(df.downfall >= downfall_borders[1]) & (df.downfall < downfall_borders[
    2])])}
            Границы интервала: [{downfall_borders[2]}; {downfall_borders[3]});\t\t3начений
    в интервале: {len(df[(df.downfall >= downfall_borders[2]) & (df.downfall < downfall_border
    s[3])])}
            Границы интервала: [{downfall_borders[3]}; {downfall_borders[4]});\t\t3начений
    в интервале: {len(df[(df.downfall >= downfall_borders[3]) & (df.downfall < downfall_border
    s[4])])}
        ''')

        plt.axvline(x=downfall_borders[1], color='#f91155')
        plt.axvline(x=downfall_borders[2], color='#f91155')
        plt.axvline(x=downfall_borders[3], color='#f91155')
```

```
592.
593.       print()
594.
595.       """__2 - Сформируйте две таблицы сопряженности, x1 и x2, x1 и x3. Постройте также м
    атрицы условных вероятностей (x1 от x2 и x1 от x3), а также таблицы коэффициентов Кетле. П
    рокомментируйте связи между категориями x1 и категориями x2 и x3__"""
596.
597.       # Таблица сопряженности x1(avg_temp) и x2(snow_depth)
598.       t_s = np.arange(16).reshape(4, 4)
599.       # Таблица сопряженности x1(avg_temp) и x3(downfall)
600.       t_d = np.arange(16).reshape(4, 4)
601.
602.       for i in range(4):
603.           for j in range(4):
604.               t_s[j][i] = len(
605.                   df[
606.                       (snow_depth_borders[i] <= df['snow_depth']) &
607.                       (df['snow_depth'] < snow_depth_borders[i+1]) &
608.                       (average_temp_borders[j] <= df['avg_temp']) &
609.                       (df['avg_temp'] < average_temp_borders[j+1])
610.                   ].index
611.               )
612.               t_d[j][i] = len(
613.                   df[
614.                       (downfall_borders[i] <= df['downfall']) &
615.                       (df['downfall'] < downfall_borders[i+1]) &
616.                       (average_temp_borders[j] <= df['avg_temp']) &
617.                       (df['avg_temp'] < average_temp_borders[j+1])
618.                   ].index
619.               )
620.
621.       print(f'''''
622.           Таблица сопряженности x1(avg_temp) и x2(snow_depth):
623.       {t_s}
624.
625.           Таблица сопряженности x1(avg_temp) и x3(downfall):
626.       {t_d}
627.       ''')
628.
629.       conditional_probability_t_s = np.zeros((4, 4))
630.       conditional_probability_t_d = np.zeros((4, 4))
631.
632.       for i in range(4):
633.           for j in range(4):
634.             conditional_probability_t_s[j,i] = round(t_s[j,i] / sum(t_s[:,i]), 2)
635.             conditional_probability_t_d[j,i] = round(t_d[j,i] / sum(t_d[:,i]), 2)
636.
637.       print(f'''''
638.           Таблица условных вероятностей x1(avg_temp) и x2(snow_depth):
639.       {conditional_probability_t_s}
640.
641.           Таблица условных вероятностей x1(avg_temp) и x3(downfall):
642.       {conditional_probability_t_d}
643.       ''')
644.
645.       probability_t_s = np.zeros((4, 4))
646.       probability_t_d = np.zeros((4, 4))
647.
648.       for i in range(4):
649.           for j in range(4):
650.             probability_t_s[j,i] = round(t_s[j,i] / t_s.sum(), 2)
651.             probability_t_d[j,i] = round(t_d[j,i] / t_d.sum(), 2)
652.
653.       ketle_t_s = np.zeros((4, 4))
654.       ketle_t_d = np.zeros((4, 4))
655.
656.       for i in range(4):
657.           for j in range(4):
```

```
            ketle_t_s[j,i] = round((probability_t_s[j,i] / ( sum(probability_t_s[:,i]) * sum(probability_t_s[j,:] )) - 1)*100, 2)
            ketle_t_d[j,i] = round((probability_t_d[j,i] / ( sum(probability_t_d[:,i]) * sum(probability_t_d[j,:] )) - 1)*100, 2)

    print(f'''''
        Таблица коэффициентов Кетле x1(avg_temp) и x2(snow_depth):
    {ketle_t_s}

        Таблица коэффициентов Кетле x1(avg_temp) и x3(downfall):
    {ketle_t_d}
    ''')

    """Как видно из таблиц, средняя глубина глубина снега растёт с уменьшением средней температуры, а осадки наиболее вероятны при температуре около 0.

    __3 - Вычислите и визуализируйте среднее значение индекса Кетле на построенных таблицах сопряженности__
    """

    ind_ketle_t_s = np.zeros((4, 4))
    ind_ketle_t_d = np.zeros((4, 4))

    for i in range(4):
        for j in range(4):
          ind_ketle_t_s[j,i] = probability_t_s[j,i]*ketle_t_s[j,i]/100
          ind_ketle_t_d[j,i] = probability_t_d[j,i]*ketle_t_s[j,i]/100

    print(f'''''
    avg ketle t-s: {ind_ketle_t_s.sum()}
    avg ketle t-d: {ind_ketle_t_d.sum()}
    ''')

    print(f'''''
    {avg_ketle_t_s}

    {avg_ketle_t_d}
    ''')

    """__4 - Прокомментируйте смысл значений индекса Кетле на двух-трех примерах__

    Для температуры больше 3.5 градусов и глубины снега больше 0 значение индекса Кетле -
100, т.е. такого не бывает, что вполне логично, так как снег при такой температуре просто растает. Для низкой температуры и глубокого снега значение индекса Кетле очень высокое, т. е. эти события почти всегда происходят одновременно.

    Для большого количества осадков и низкой температуры индекс Кетле очень низкий, а для температуры около 0 и осадков индекс Кетле высокий, т.е. осадки наиболее вероятны при температуре у 0, и крайне маловероятны при морозах.

    # Задание 5: МГК/SVD

     1. Выберите в Ваших данных 3-6 признаков, более или менее относящихся к одному и тому же аспекту данных; откомментируйте
     2. Визуализируйте Ваши данные дважды, один раз с использованием стандартизации на размах, второй – путем z-scoring. Выберите какую-либо группу объектов и выделите ее на визуализациях цветом или формой «точек»
     3. Примените традиционный метод МГК для визуализации (для какого-либо способа нормализации) и убедитесь, что получено то же самое, что и выше. Если нет – объясните, в чем дело
     4. Постарайтесь проинтерпретировать полученные компоненты

    __1 - Выберите в Ваших данных 3-6 признаков, более или менее относящихся к одному и тому же аспекту данных; откомментируйте__
    """
```

```
709.        av_t = df[['avg_temp', 'min_temp', 'max_temp']].to_numpy()
710.        amplitude_scoring = pd.DataFrame((av_t - av_t.mean(axis = 0))/(av_t.max(axis = 0) -
    av_t.min(axis = 0)), columns = ['avg_temp', 'min_temp', 'max_temp'])
711.        amplitude_scoring['snow_depth'] = df['snow_depth'].to_numpy()
712.        z_scoring = pd.DataFrame((av_t - av_t.mean(axis = 0))/np.sqrt( ((av_t - av_t.mean(a
    xis = 0))*(av_t - av_t.mean(axis = 0))).mean(axis = 0) ), columns = ['avg_temp', 'min_temp
    ', 'max_temp'])
713.        z_scoring['snow_depth'] = df['snow_depth'].to_numpy()
714.
715.        """__2 - Визуализируйте Ваши данные дважды, один раз с использованием стандартизаци
    и на размах, второй – путем z-scoring. Выберите какую-
    либо группу объектов и выделите ее на визуализациях цветом или формой «точек»__"""
716.
717.        print('Визуализируем данные с использованием стандартизации на размах:\n')
718.        fig, graphs = plt.subplots(1, 3, figsize=(17.5, 5))
719.        plt.axes(graphs[0])
720.        plt.scatter(
721.                amplitude_scoring[amplitude_scoring['snow_depth'] == 0]['avg_temp'],
722.                amplitude_scoring[amplitude_scoring['snow_depth'] == 0]['min_temp'],
723.                label='no_snow',
724.                cmap='rainbow'
725.            )
726.        plt.scatter(
727.                amplitude_scoring[amplitude_scoring['snow_depth'] > 0]['avg_temp'],
728.                amplitude_scoring[amplitude_scoring['snow_depth'] > 0]['min_temp'],
729.                label='snow',
730.                cmap='rainbow'
731.            )
732.        plt.xlabel('avg_temp')
733.        plt.ylabel('min_temp')
734.        plt.legend(title='is_snow')
735.
736.        plt.axes(graphs[1])
737.        plt.scatter(
738.                amplitude_scoring[amplitude_scoring['snow_depth'] == 0]['avg_temp'],
739.                amplitude_scoring[amplitude_scoring['snow_depth'] == 0]['max_temp'],
740.                label='no_snow',
741.                cmap='rainbow'
742.            )
743.        plt.scatter(
744.                amplitude_scoring[amplitude_scoring['snow_depth'] > 0]['avg_temp'],
745.                amplitude_scoring[amplitude_scoring['snow_depth'] > 0]['max_temp'],
746.                label='snow',
747.                cmap='rainbow'
748.            )
749.        plt.xlabel('avg_temp')
750.        plt.ylabel('max_temp')
751.        plt.legend(title='is_snow')
752.
753.        plt.axes(graphs[2])
754.        plt.scatter(
755.                amplitude_scoring[amplitude_scoring['snow_depth'] == 0]['min_temp'],
756.                amplitude_scoring[amplitude_scoring['snow_depth'] == 0]['max_temp'],
757.                label='no_snow',
758.                cmap='rainbow'
759.            )
760.        plt.scatter(
761.                amplitude_scoring[amplitude_scoring['snow_depth'] > 0]['min_temp'],
762.                amplitude_scoring[amplitude_scoring['snow_depth'] > 0]['max_temp'],
763.                label='no_snow',
764.                cmap='rainbow'
765.            )
766.        plt.xlabel('min_temp')
767.        plt.ylabel('max_temp')
768.        plt.legend(title='is_snow')
769.
770.        from sklearn.decomposition import PCA
771.
772.        pca_a = PCA(n_components = 2)
```

```
773.        XPCAreduced = pca_a.fit_transform(amplitude_scoring[['avg_temp', 'min_temp', 'max_t
    emp']].to_numpy())
774.        a_s = pd.DataFrame(XPCAreduced, columns = ['col1', 'col2'])
775.        a_s['snow_depth'] = df['snow_depth'].to_numpy()
776.
777.        fig, graph = plt.subplots(figsize=(10, 6))
778.        plt.scatter(
779.                a_s[a_s['snow_depth'] == 0]['col1'],
780.                a_s[a_s['snow_depth'] == 0]['col2'],
781.                label='Нет снега',
782.                cmap='rainbow'
783.            )
784.        plt.scatter(
785.                a_s[a_s['snow_depth'] > 0]['col1'],
786.                a_s[a_s['snow_depth'] > 0]['col2'],
787.                label='Есть снег',
788.                cmap='rainbow'
789.            )
790.        plt.title('Визуализация данных с использованием МГК и стандартизации на размах')
791.        plt.xlabel('dim_1')
792.        plt.ylabel('dim_2')
793.        plt.grid()
794.        plt.legend(title='Снег')
795.
796.        print()
797.
798.        fig, graphs = plt.subplots(1, 3, figsize=(17.5, 5))
799.        fig.suptitle('Визуализация данных с использованием z-scoring', fontsize=16)
800.
801.        plt.axes(graphs[0])
802.        plt.scatter(
803.                z_scoring[z_scoring['snow_depth'] == 0]['avg_temp'],
804.                z_scoring[z_scoring['snow_depth'] == 0]['min_temp'],
805.                label='Нет снега',
806.                cmap='rainbow'
807.            )
808.        plt.scatter(
809.                z_scoring[z_scoring['snow_depth'] > 0]['avg_temp'],
810.                z_scoring[z_scoring['snow_depth'] > 0]['min_temp'],
811.                label='Есть снег',
812.                cmap='rainbow'
813.            )
814.        plt.xlabel('avg_temp')
815.        plt.ylabel('min_temp')
816.        plt.legend(title='Снег')
817.
818.        plt.axes(graphs[1])
819.        plt.scatter(
820.                z_scoring[z_scoring['snow_depth'] == 0]['avg_temp'],
821.                z_scoring[z_scoring['snow_depth'] == 0]['max_temp'],
822.                label='Нет снега',
823.                cmap='rainbow'
824.            )
825.        plt.scatter(
826.                z_scoring[z_scoring['snow_depth'] > 0]['avg_temp'],
827.                z_scoring[z_scoring['snow_depth'] > 0]['max_temp'],
828.                label='Есть снег',
829.                cmap='rainbow'
830.            )
831.        plt.xlabel('avg_temp')
832.        plt.ylabel('max_temp')
833.        plt.legend(title='Снег')
834.
835.        plt.axes(graphs[2])
836.        plt.scatter(
837.                z_scoring[z_scoring['snow_depth'] == 0]['min_temp'],
838.                z_scoring[z_scoring['snow_depth'] == 0]['max_temp'],
839.                label='Нет снега',
840.                cmap='rainbow'
```

```
            )
        plt.scatter(
                z_scoring[z_scoring['snow_depth'] > 0]['min_temp'],
                z_scoring[z_scoring['snow_depth'] > 0]['max_temp'],
                label='Есть снег',
                cmap='rainbow'
            )
        plt.xlabel('min_temp')
        plt.ylabel('max_temp')
        plt.legend(title='Снег')
        fig.savefig('gdrive/My Drive/Colab Notebooks/task5_zscoring.png')

        print()

        from sklearn.decomposition import PCA

        pca_z = PCA(n_components = 2)
        XPCAreduced = pca_z.fit_transform(z_scoring[['avg_temp', 'min_temp', 'max_temp']].to_numpy())
        z_s = pd.DataFrame(XPCAreduced, columns = ['col1', 'col2'])
        z_s['snow_depth'] = df['snow_depth'].to_numpy()

        fig, graph = plt.subplots(figsize=(10, 6))
        plt.scatter(
                z_s[z_s['snow_depth'] == 0]['col1'],
                z_s[z_s['snow_depth'] == 0]['col2'],
                label='Нет снега',
                cmap='rainbow'
            )
        plt.scatter(
                z_s[z_s['snow_depth'] > 0]['col1'],
                z_s[z_s['snow_depth'] > 0]['col2'],
                label='Снег',
                cmap='rainbow'
            )

        plt.title('Визуализация данных с использованием МГК и z-scoring')
        plt.xlabel('dim_1')
        plt.ylabel('dim_2')
        plt.grid()
        plt.legend(title='Снег')

        print()

        X = np.transpose(df[['avg_temp', 'min_temp', 'max_temp']].to_numpy())
        Xcentered = (X[0] - X[0].mean(), X[1] - X[1].mean(), X[2] - X[2].mean())
        m = (X[0].mean(), X[1].mean(), X[2].mean())
        covmat = np.cov(Xcentered)
        _, vecs = np.linalg.eig(covmat)
        v = [vecs[:,0], vecs[:,2]]
        Xnew = pd.DataFrame(np.transpose(np.dot(v,Xcentered)), columns = ['col1', 'col2'])

        Xnew['snow_depth'] = df['snow_depth'].to_numpy()

        fig, graph = plt.subplots(figsize=(10, 6))
        plt.scatter(
                Xnew[Xnew['snow_depth'] == 0]['col1'],
                Xnew[Xnew['snow_depth'] == 0]['col2'],
                label='Нет снега',
                cmap='rainbow'
            )
        plt.scatter(
                Xnew[Xnew['snow_depth'] > 0]['col1'],
                Xnew[Xnew['snow_depth'] > 0]['col2'],
                label='Есть снег',
                cmap='rainbow'
            )

```

```
907.       plt.title('Визуализация данных с использованием традиционного МГК, реализованного в
    ручную')
908.       plt.legend(title='Снег')
909.       plt.xlabel('dim_1')
910.       plt.ylabel('dim_2')
911.       plt.grid()
912.
913.       print()
914.
915.       print ("Первая главная компонента: ", v[0], "\nВторая главная компонента: ", v[1])

916.
917.       """В первой главной компоненте коэффициенты почти равны, таким образом первая компо
    нента это некий общий показатель температуры, на основе всех трёх температурных признаков.

918.
919.       Во второй главной компоненте коэффициент перед среднесуточной температурой практиче
    ски равен 0, а коэффициенты перед минимальной и максимальной температурой почти равны по м
    одулю и противоположны по знаку, значит это разность минимальной и максимальной температур
    ы (домноженная на какой-
    то коэффициент), т.е. амплитуда колебания температуры в течении суток.
920.
921.       # Задание 6 - Линейная регрессия
922.
923.        * По возможности, найдите два признака в Ваших данных с более или менее «линейным»
     полем рассеяния; представьте его на графике.
924.        * Постройте уравнение линейной регрессии одного из этих признаков через другой при
    знак. Сделайте комментарий о смысле величины регрессионного коэффициента при этом другом п
    ризнаке.
925.        * Найдите значения коэффициентов корреляции и детерминации; сделайте комментарий о
     смысле величины последнего.
926.        * Сделайте предсказание величины целевого признака на двух-
    трех объектах; прокомментируйте результат.
927.        * Рассчитайте среднюю относительную ошибку регрессионного уравнения на всех объект
    ах Вашей таблицы данных и
928.       сравните ее с величиной коэффициента детерминации.
929.       """
930.
931.       import seaborn as sns
932.
933.       sns.set()
934.       cols = ['date','avg_temp','min_temp','max_temp','downfall','pressure','humidity','w
    ind_speed','snow_depth','year']
935.       sns.pairplot(df[cols])
936.
937.       fig = plt.gcf()
938.       fig.savefig('gdrive/My Drive/Colab Notebooks/task6_pairplot.png')
939.       print()
940.
941.       """Более остальных на линейное поле рассеивания похоже поле рассеивания признаков `
    date` и `avg_temp`"""
942.
943.       fig, graph = plt.subplots(figsize=(13, 9))
944.       for i in range(1, 31):
945.         plt.scatter(
946.               df[df.date == i].date,
947.               df[df.date == i].avg_temp,
948.               label=f'{i} ноября',
949.               cmap='rainbow'
950.           )
951.       plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
952.       plt.tight_layout()
953.       plt.title('Поле рассеивания признаков avg_temp и date')
954.       plt.xlabel('date')
955.       plt.ylabel('avg_temp')
956.
957.       fig.savefig('gdrive/My Drive/Colab Notebooks/task6_scatter_plot.png')
958.
959.       print()
```

```
960.
961.       """Обучим линейную регрессию:
962.        * `date` - признак
963.        * `avg_temp` - целевое значение
964.       """
965.
966.       from sklearn.linear_model import LinearRegression
967.
968.       model = LinearRegression()
969.       model.fit(np.array(df['date'])[:,np.newaxis], np.array(df['avg_temp'])[:,np.newaxis
    ])
970.
971.       print(f'Уравнение линейной регрессии имеет вид:\ty = {model.coef_.item()} * x + {mo
    del.intercept_.item()}')
972.
973.       fig, graph = plt.subplots(figsize=(13, 9))
974.       for i in range(1, 31):
975.         plt.scatter(
976.               df[df.date == i].date,
977.               df[df.date == i].avg_temp,
978.               label=f'{i} ноября',
979.               cmap='rainbow'
980.           )
981.
982.       x = np.linspace(0,30)
983.       y = model.coef_.item()*x + model.intercept_.item()
984.       plt.plot(x, y, label='Уравнение линейной регрессии')
985.       plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
986.       plt.tight_layout()
987.       plt.title('Поле рассеивания признаков avg_temp и date')
988.       plt.xlabel('date')
989.       plt.ylabel('avg_temp')
990.
991.       fig.savefig('gdrive/My Drive/Colab Notebooks/task6_scatter_plot.png')
992.
993.       print()
994.
995.       from scipy.stats import pearsonr
996.       from sklearn.metrics import r2_score
997.
998.       corr = pearsonr(model.coef_.item()*df['date'] + model.intercept_.item(), df['avg_te
    mp'])[0]
999.       r2 = r2_score(df['avg_temp'], model.coef_.item()*df['date'] + model.intercept_.item
    ())
1000.
1001.      print(f'Значение коэффициента корреляции Пирсона:\t{corr}')
1002.      print(f'Значение коэффициента детерминации:\t\t{r2}')
1003.
1004.      """Сделаем интересное наблюдение, а именно - проверим, какую погоду температуру мод
    ель на нескольких числах ноября и сравним её с действительной за окном;"""
1005.
1006.      print(f'''''
1007.      1 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): -
    0.5˚\tПредсказание:\t{model.predict(np.array([1])[:,np.newaxis]).item()}˚
1008.      8 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): +3˚\tПредсказание:\t
    {model.predict(np.array([8])[:,np.newaxis]).item()}˚
1009.      15 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): +3.5˚\tПредсказание
    :\t{model.predict(np.array([15])[:,np.newaxis]).item()}˚
1010.      22 ноября 2019\tСредняя дневная температура (по Яндекс.Погода): -
    3.5˚\tПредсказание:\t{model.predict(np.array([22])[:,np.newaxis]).item()}
1011.      ''')
1012.
1013.      """Не то, чтобы прямо точно, но мы и не синоптики в конце концов. А по данным модел
    и, в целом, тренд прослеживается, и примерно понятно, когда стоит надеть свитер :)"""
1014.
1015.      tmp = df[['date', 'avg_temp']]
1016.      tmp['avg_temp_predict'] = tmp.apply(lambda x: x.date * model.coef_.item() + model.i
    ntercept_.item(), axis=1)
```

```
1017.      tmp['diff_percentage'] = tmp.apply(lambda x: abs((x.avg_temp - x.avg_temp_predict)
    * 100.0 / x.avg_temp), axis=1)
1018.      print(f'Значение средней относительной ошибки: {tmp.diff_percentage.mean()}%')
```