**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.

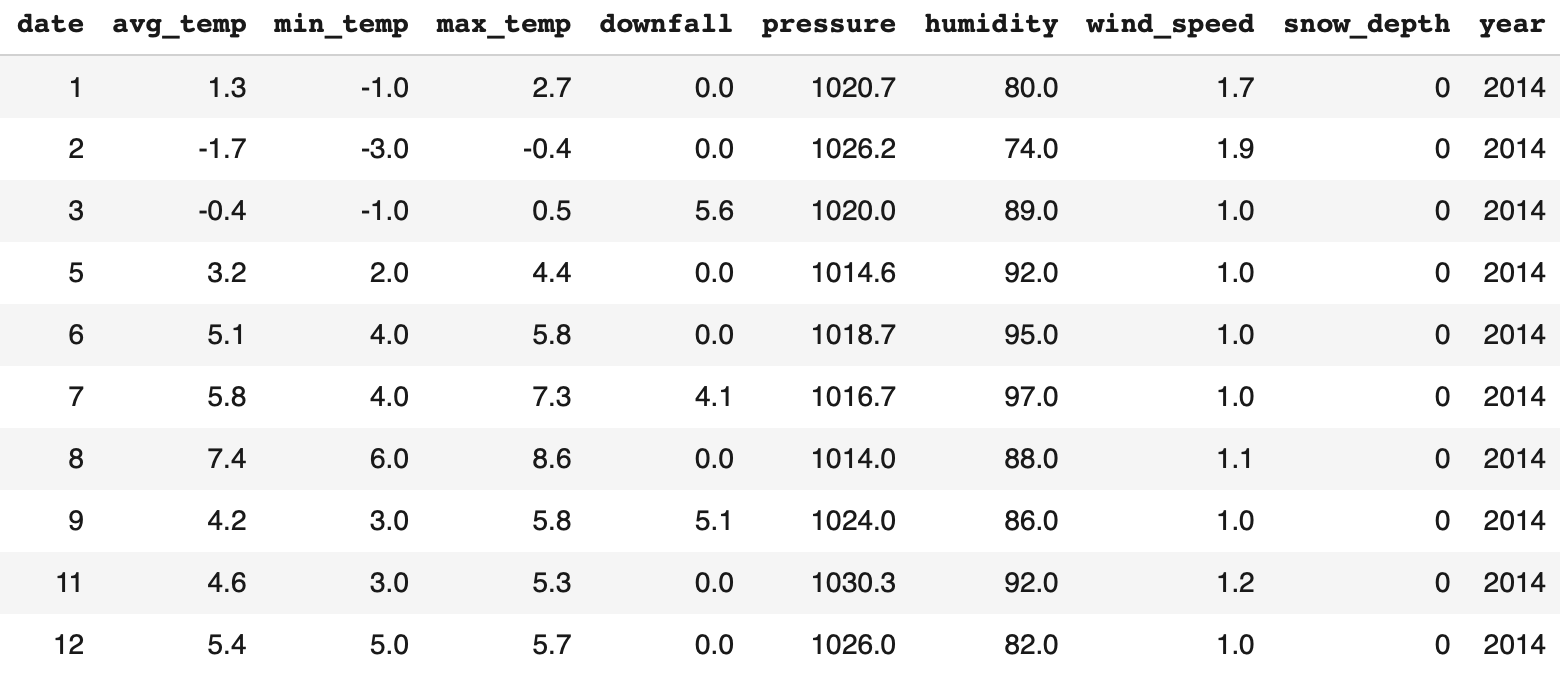


Таблица 1 - Пример данных из датасета

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
3. # Выбираем нужные нам признаки из общего датасета
4. df\_features = df[['avg\_temp', 'pressure', 'wind\_speed']]
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).

Изображение выглядит как снимок экрана

Автоматически созданное описание

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

Теперь выполним аналогичные действия для k=9:

1. from sklearn.cluster import KMeans
3. # Выбираем нужные нам признаки из общего датасета
4. df\_features = df[['avg\_temp', 'pressure', 'wind\_speed']]
6. kmeans\_9 = KMeans(
7. n\_clusters = 9,       # 9 кластеров
8. init = 'random',      # центры кластеров - случайно выбранные точки из датасета
9. n\_init = 10           # количество инициализаций алгоритма
10. )
11. kmeans\_9.fit(df\_features) # выполнение алгоритма

Изображение выглядит как снимок экрана

Автоматически созданное описание

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:

Изображение выглядит как снимок экрана

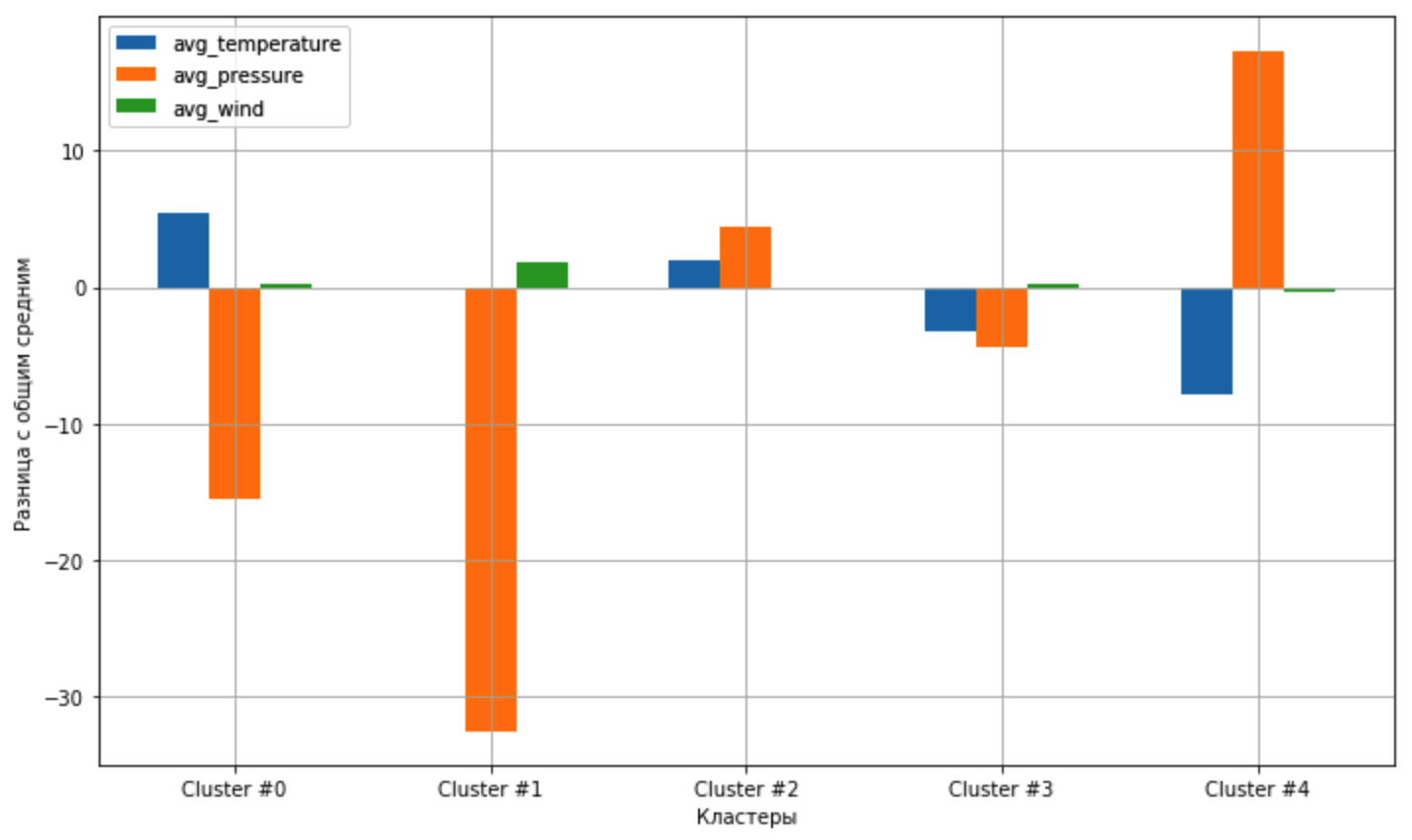
Автоматически созданное описание 

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.

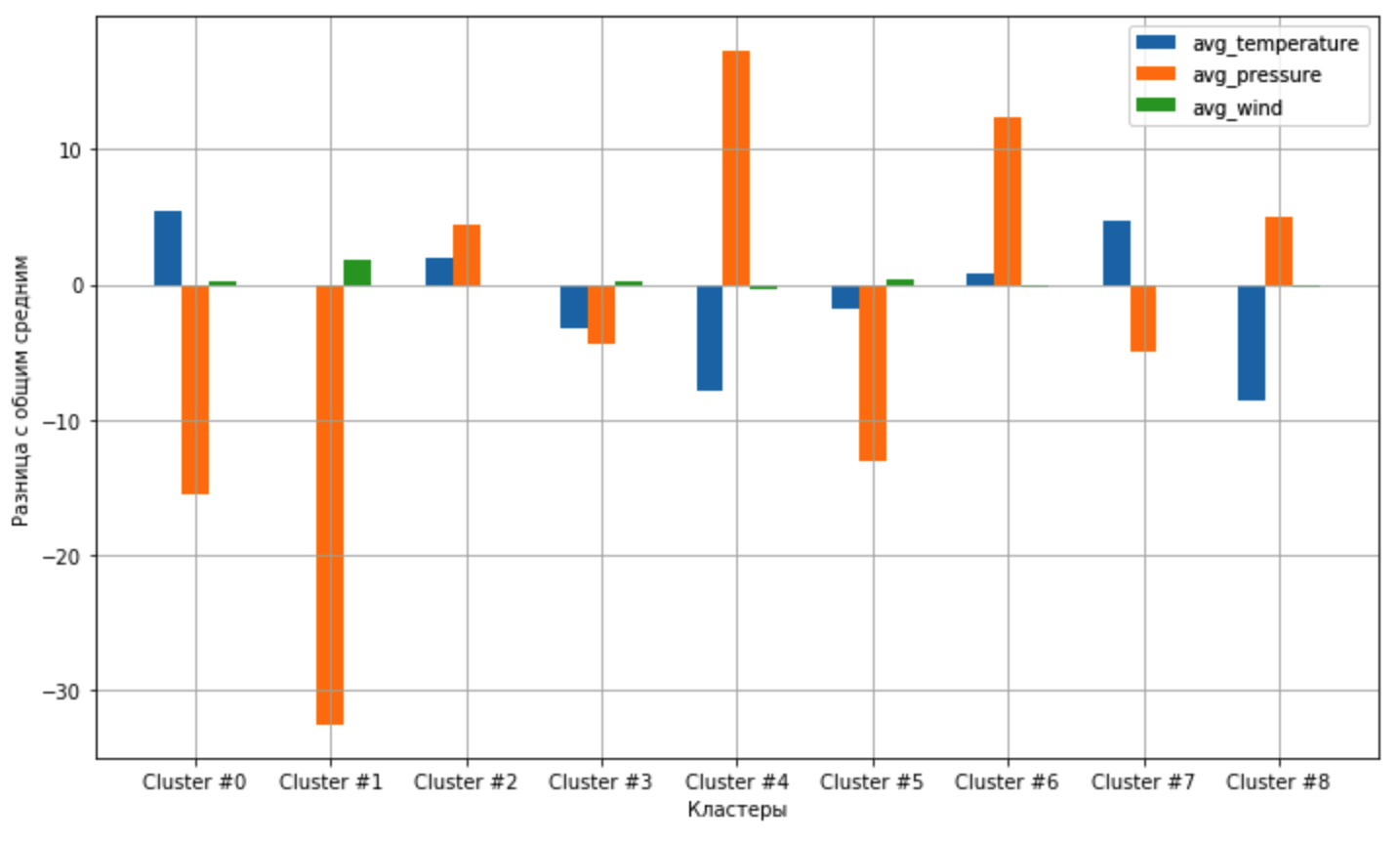
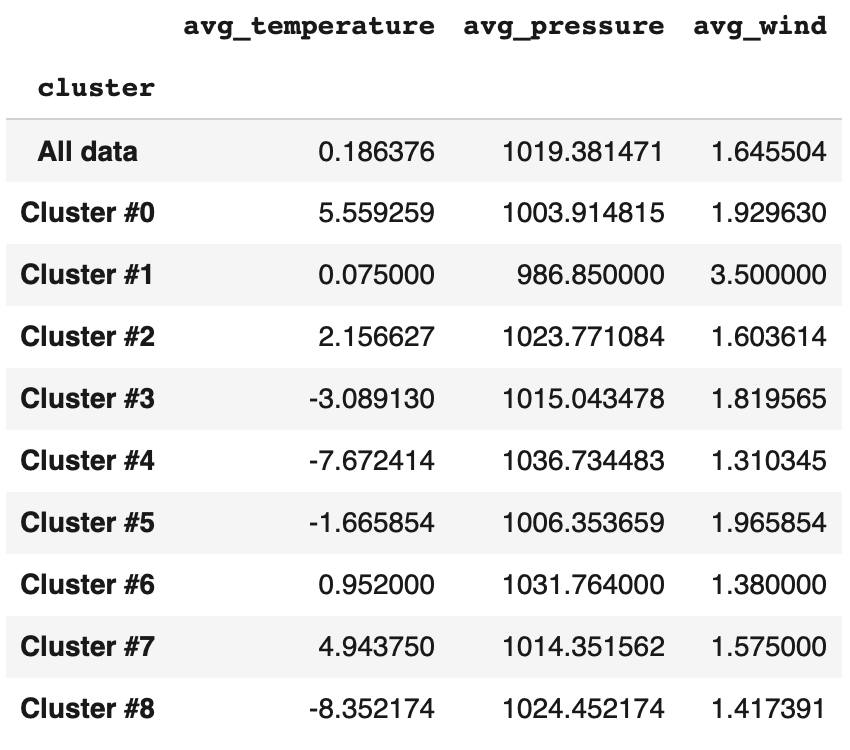


Иллюстрация 4 - Таблица средних и bar chart разницы внутрикластерного и общего средних для разбиения на 9 кластеров

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:

1. **from** random **import** choices
2. **from** statistics **import** mean
4. # Список средних
5. means = []
7. # Количество экспериментов
8. bootstrap\_iters = 5000
9. # Количество выбираемых значений для эксперимента
10. bootstrap\_choices = len(df\_features)
12. **for** i **in** range(bootstrap\_iters):
13. # Добавляем в список
14. means.append(
15. # Среднее значение
16. mean(
17. choices(
18. # Выбираем случайный элемент из значений признака
19. df\_features\_5['avg\_temp'].tolist(),
20. # k-раз
21. k=bootstrap\_choices
22. )
23. )
24. )
26. # Визуализация
27. fig, graph = plt.subplots(figsize=(10, 6))
28. plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')

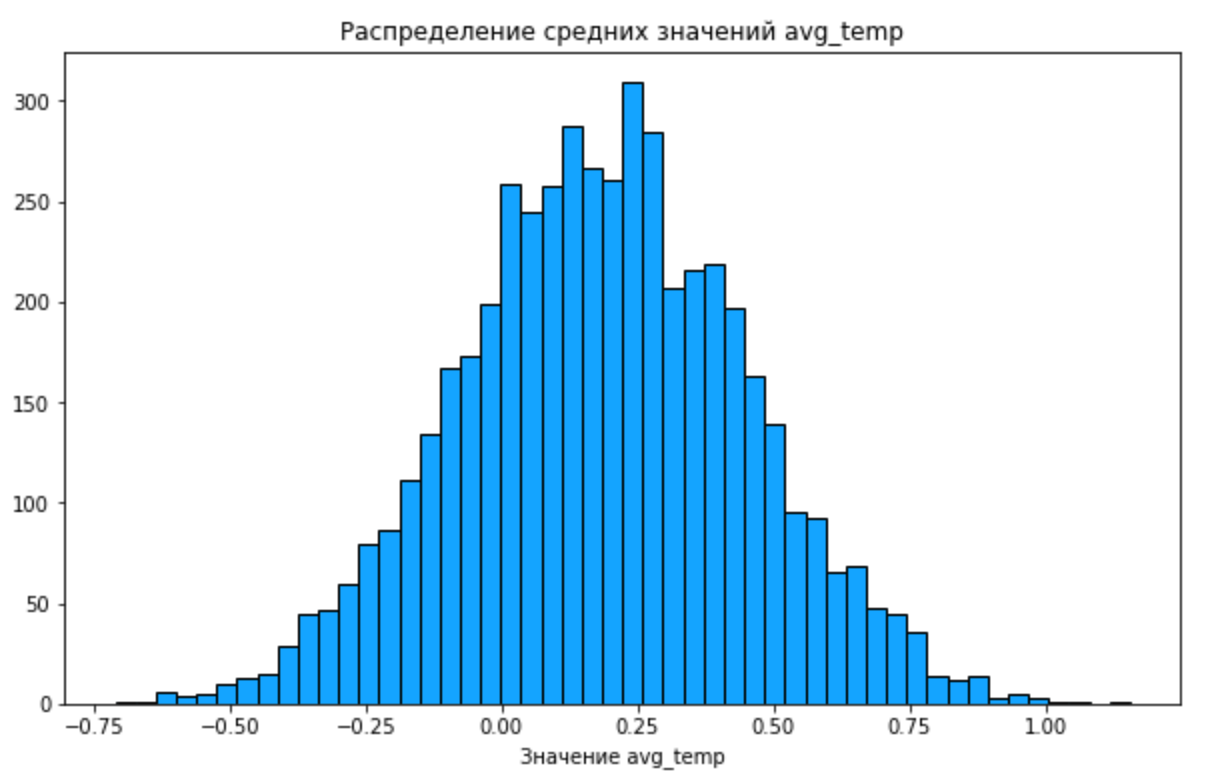


Иллюстрация 5 - Распределение средних значений avg\_temp

Найдем 95% доверительный интервал способом "без опоры":

1. # Сортируем список со средними
2. means\_ordered = sorted(means)
3. # Отсекаем 2.5% значений снизу и берем наименьший оставшийся
4. left = means\_ordered[int(len(means\_ordered)\*0.05/2 - 1)]
5. # Отсекаем 2.5% значений сверху и берем наибольший оставшийся
6. right = means\_ordered[int(len(means\_ordered) - (len(means\_ordered)\*0.05/2) - 1)]
8. **print**(f'''''
9. Границы 95% доверительного интервала:
10. \tЛевая:\t{left}
11. \tПравая:\t{right}
12. ''')
14. >>>Границы 95% доверительного интервала:
15. >>>    Левая:  -0.34005449591280656
16. >>>    Правая: 0.6956403269754768

Найдем 95% доверительный интервал способом "c опорой":

1. **from** statistics **import** mean, stdev
3. **print**(f'''''
4. Границы 95% доверительного интервала:
5. \tЛевая:\t{mean(means) - 1.96\*stdev(means)}
6. \tПравая:\t{mean(means) + 1.96\*stdev(means)}
7. ''')
9. >>>Границы 95% доверительного интервала:
10. >>>    Левая:  -0.33938573245791703
11. >>>    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:

1. means = []
2. bootstrap\_iters = 5000
4. **for** i **in** range(bootstrap\_iters):
5. means.append(
6. # Из среднего значения признака в эксперименте на 3 кластере
7. mean(
8. choices(
9. df\_features\_5[df\_features\_5.cluster == 3]['avg\_temp'].tolist(),
10. k=len(df\_features\_5[df\_features\_5.cluster == 3])
11. )
12. )
13. # вычитаем
14. -
15. # среднее значение признака в эксперименте на 4 кластере
16. mean(
17. choices(
18. df\_features\_5[df\_features\_5.cluster == 4]['avg\_temp'].tolist(),
19. k=len(df\_features\_5[df\_features\_5.cluster == 4])
20. )
21. )
22. )
24. # Визуализация
25. fig, graph = plt.subplots(figsize=(10, 6))
26. plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')

Изображение выглядит как музыка

Автоматически созданное описание

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.

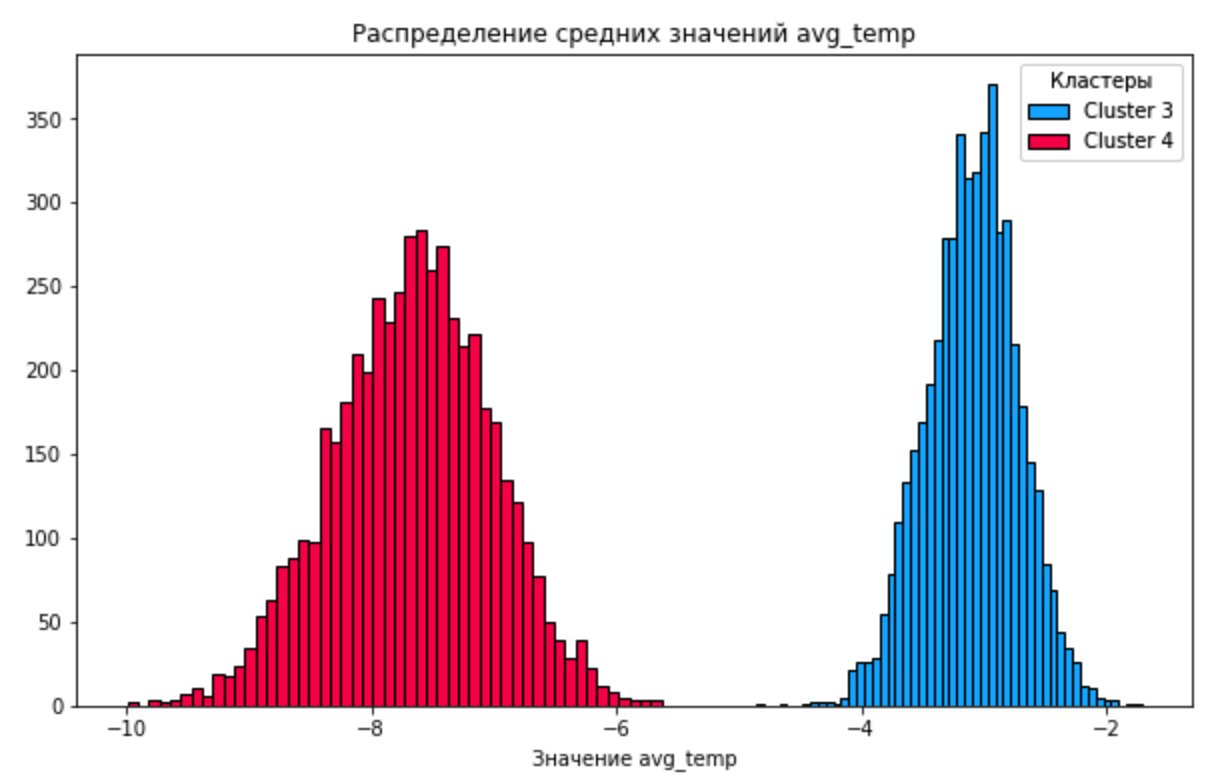


Иллюстрация 7 - Распределение средних внутрикластерных значений признака avg\_temp в кластерах №3 и №4

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

1. >>>Границы 95% доверительного интервала (без опоры):
2. >>>    Левая:  3.112893553223388
3. >>>    Правая: 6.068665667166417
4. >>>
5. >>>Границы 95% доверительного интервала (с опорой):
6. >>>    Левая:  3.1109550134891464
7. >>>    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:

1. means = []
3. bootstrap\_iters = 5000
5. **for** i **in** range(bootstrap\_iters):
6. means.append(
7. mean(
8. choices(
9. df\_features\_5[df\_features\_5.cluster == 1]['avg\_temp'].tolist(),
10. k=len(df\_features\_5[df\_features\_5.cluster == 1])
11. )
12. )
13. -
14. mean(
15. choices(
16. df\_features\_5['avg\_temp'].tolist(),
17. k=len(df\_features\_5)
18. )
19. )
20. )
22. fig, graph = plt.subplots(figsize=(10, 6))
23. plt.hist(means, bins=50, color='#00b5ff', edgecolor='black')

Изображение выглядит как музыка

Автоматически созданное описание

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.

Изображение выглядит как снимок экрана

Автоматически созданное описание

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:

1. >>>Границы 95% доверительного интервала:
2. >>>    Левая:  -3.8915531335149858
3. >>>    Правая: 3.684264305177112
4. >>>
5. >>>Границы 95% доверительного интервала:
6. >>>    Левая:  -3.9516607523396794
7. >>>    Правая: 3.747105384492268

ТАБЛИЦА СОПРЯЖЕННОСТИ

Сформируем три номинальных признака х1, х2 и х3. В качестве первого номинального признака возьмем разбиение диапазона значений признака avg\_temp на 4 примерно равных по количеству объектов интервала.

1. # Разбиение
2. temp\_sorted = df['avg\_temp'].sort\_values()
3. average\_temp\_borders =[
4. temp\_sorted.iloc[0],
5. temp\_sorted.iloc[len(temp\_sorted.index) // 4],
6. temp\_sorted.iloc[(len(temp\_sorted.index) // 4) \* 2],
7. temp\_sorted.iloc[(len(temp\_sorted.index) // 4) \* 3],
8. temp\_sorted.iloc[len(temp\_sorted.index)-1] + 0.1
9. ]
10. # Визуализация
11. fig, graph = plt.subplots(figsize=(10, 6))
12. plt.hist(df['avg\_temp'], bins=15, color='#00b5ff', label='avg\_temp', edgecolor='black')
13. plt.axvline(x=average\_temp\_borders[1], color='#f91155')
14. plt.axvline(x=average\_temp\_borders[2], color='#f91155')
15. plt.axvline(x=average\_temp\_borders[3], color='#f91155')
16. # Вывод
17. **print**(f'''''
18. 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])])}
19. 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])])}
20. 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])])}
21. 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])])}
22. ''')

Изображение выглядит как снимок экрана

Автоматически созданное описание

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:

1. fig, graph = plt.subplots(figsize=(10, 6))
2. plt.hist(df['snow\_depth'], bins=30, color='#00b5ff', label='snow\_depth', edgecolor='black')
4. snow\_depth\_borders = [0, 0.1, 5, 13, 27]
5. **print**(f'''''Рассмотрим распределение, и выберем для него границы интервалов
6. 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])])}
7. 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])])}
8. 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])])}
9. 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])])}
10. ''')
12. plt.axvline(x=snow\_depth\_borders[1], color='#f91155')
13. plt.axvline(x=snow\_depth\_borders[2], color='#f91155')
14. plt.axvline(x=snow\_depth\_borders[3], color='#f91155')

Изображение выглядит как снимок экрана

Автоматически созданное описание

Иллюстрация 11 - Разбиение объектов по признаку snow\_depth на 4 интервала

Для выделения третьего номинального признака рассмотрим распределение признака downfall:

1. fig, graph = plt.subplots(figsize=(10, 6))
2. plt.hist(df['downfall'], bins=30, color='#00b5ff', label='downfall', edgecolor='black')
4. downfall\_borders = [0, 0.1, 4, 7, 28]
5. **print**(f'''''Рассмотрим распределение, и выберем для него границы интервалов
6. Границы интервала: [{downfall\_borders[0]}; {downfall\_borders[1]});\tЗначений в интервале: {len(df[(df.downfall >= downfall\_borders[0]) & (df.downfall < downfall\_borders[1])])}
7. Границы интервала: [{downfall\_borders[1]}; {downfall\_borders[2]});\tЗначений в интервале: {len(df[(df.downfall >= downfall\_borders[1]) & (df.downfall < downfall\_borders[2])])}
8. Границы интервала: [{downfall\_borders[2]}; {downfall\_borders[3]});\t\tЗначений в интервале: {len(df[(df.downfall >= downfall\_borders[2]) & (df.downfall < downfall\_borders[3])])}
9. Границы интервала: [{downfall\_borders[3]}; {downfall\_borders[4]});\t\tЗначений в интервале: {len(df[(df.downfall >= downfall\_borders[3]) & (df.downfall < downfall\_borders[4])])}
10. ''')
12. plt.axvline(x=downfall\_borders[1], color='#f91155')
13. plt.axvline(x=downfall\_borders[2], color='#f91155')
14. plt.axvline(x=downfall\_borders[3], color='#f91155')

Изображение выглядит как снимок экрана

Автоматически созданное описание

Иллюстрация 12 - Разбиение объектов по признаку downfall на 4 интервала

Сформируем две таблицы сопряженности, х1 и х2, х1 и х3:

1. # Таблица сопряженности x1(avg\_temp) и x2(snow\_depth)
2. t\_s = np.arange(16).reshape(4, 4)
3. # Таблица сопряженности x1(avg\_temp) и x3(downfall)
4. t\_d = np.arange(16).reshape(4, 4)
6. **for** i **in** range(4):
7. **for** j **in** range(4):
8. t\_s[j][i] = len(
9. df[
10. (snow\_depth\_borders[i] <= df['snow\_depth']) &
11. (df['snow\_depth'] < snow\_depth\_borders[i+1]) &
12. (average\_temp\_borders[j] <= df['avg\_temp']) &
13. (df['avg\_temp'] < average\_temp\_borders[j+1])
14. ].index
15. )
16. t\_d[j][i] = len(
17. df[
18. (downfall\_borders[i] <= df['downfall']) &
19. (df['downfall'] < downfall\_borders[i+1]) &
20. (average\_temp\_borders[j] <= df['avg\_temp']) &
21. (df['avg\_temp'] < average\_temp\_borders[j+1])
22. ].index
23. )

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 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 (х1 от х2 и х1 от х3):

1. conditional\_probability\_t\_s = np.zeros((4, 4))
2. conditional\_probability\_t\_d = np.zeros((4, 4))
4. **for** i **in** range(4):
5. **for** j **in** range(4):
6. conditional\_probability\_t\_s[j,i] = round(t\_s[j,i] / sum(t\_s[:,i]), 2)
7. 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:

1. probability\_t\_s = np.zeros((4, 4))
2. probability\_t\_d = np.zeros((4, 4))
4. **for** i **in** range(4):
5. **for** j **in** range(4):
6. probability\_t\_s[j,i] = round(t\_s[j,i] / t\_s.sum(), 2)
7. probability\_t\_d[j,i] = round(t\_d[j,i] / t\_d.sum(), 2)
9. ketle\_t\_s = np.zeros((4, 4))
10. ketle\_t\_d = np.zeros((4, 4))
12. **for** i **in** range(4):
13. **for** j **in** range(4):
14. ketle\_t\_s[j,i] = round((probability\_t\_s[j,i] / ( sum(probability\_t\_s[:,i]) \* sum(probability\_t\_s[j,:] )) - 1)\*100, 2)
15. ketle\_t\_d[j,i] = round((probability\_t\_d[j,i] / ( sum(probability\_t\_d[:,i]) \* sum(probability\_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:

1. ind\_ketle\_t\_s = np.zeros((4, 4))
2. ind\_ketle\_t\_d = np.zeros((4, 4))
4. **for** i **in** range(4):
5. **for** j **in** range(4):
6. ind\_ketle\_t\_s[j,i] = probability\_t\_s[j,i]\*ketle\_t\_s[j,i]/100
7. ind\_ketle\_t\_d[j,i] = probability\_t\_d[j,i]\*ketle\_t\_s[j,i]/100
9. **print**(f'''''
10. Cреднее значение индекса Кетле x1(avg\_temp) и x2(snow\_depth): {ind\_ketle\_t\_s.sum()}
11. Cреднее значение индекса Кетле x1(avg\_temp) и x3(downfall): {ind\_ketle\_t\_d.sum()}
12. ''')

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | snow\_depth | | | | |
| **[0, 0.1)** | **[0.1, 5)** | **[5, 13)** | **[13, 27)** | **Total** |
| avg\_temp | **[-16.9, -2.2)** | 0,0017 | 0,1019 | 0,0972 | 0,2306 | 0,4313 |
| **[-2.2, 1.0)** | 0,0517 | 0,0170 | -0,0016 | 0,2112 | 0,2783 |
| **[1.0, 3.6)** | 0,0044 | -0,0094 | -0,0022 | 0,2356 | 0,2285 |
| **[3.6, 14.0)** | 0,0000 | 0,0000 | 0,0000 | 0,2614 | 0,2614 |
| **Total** | 0,0578 | 0,1095 | 0,0933 | 0,9389 | 1,1995 |

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», являются 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.min(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:

1. **from** sklearn.decomposition **import** PCA
3. pca\_a = PCA(n\_components = 2)
4. XPCAreduced = pca\_a.fit\_transform(amplitude\_scoring[['avg\_temp', 'min\_temp', 'max\_temp']].to\_numpy())
5. a\_s = pd.DataFrame(XPCAreduced, columns = ['col1', 'col2'])
6. a\_s['snow\_depth'] = df['snow\_depth'].to\_numpy()
8. fig, graph = plt.subplots(figsize=(10, 6))
9. plt.scatter(
10. a\_s[a\_s['snow\_depth'] == 0]['col1'],
11. a\_s[a\_s['snow\_depth'] == 0]['col2'],
12. label='Нет снега',
13. cmap='rainbow'
14. )
15. plt.scatter(
16. a\_s[a\_s['snow\_depth'] > 0]['col1'],
17. a\_s[a\_s['snow\_depth'] > 0]['col2'],
18. label='Есть снег',
19. cmap='rainbow'
20. )
21. plt.title('Визуализация данных с использованием МГК и стандартизации на размах')
22. plt.xlabel('dim\_1')
23. plt.ylabel('dim\_2')
24. plt.grid()
25. plt.legend(title='Снег')

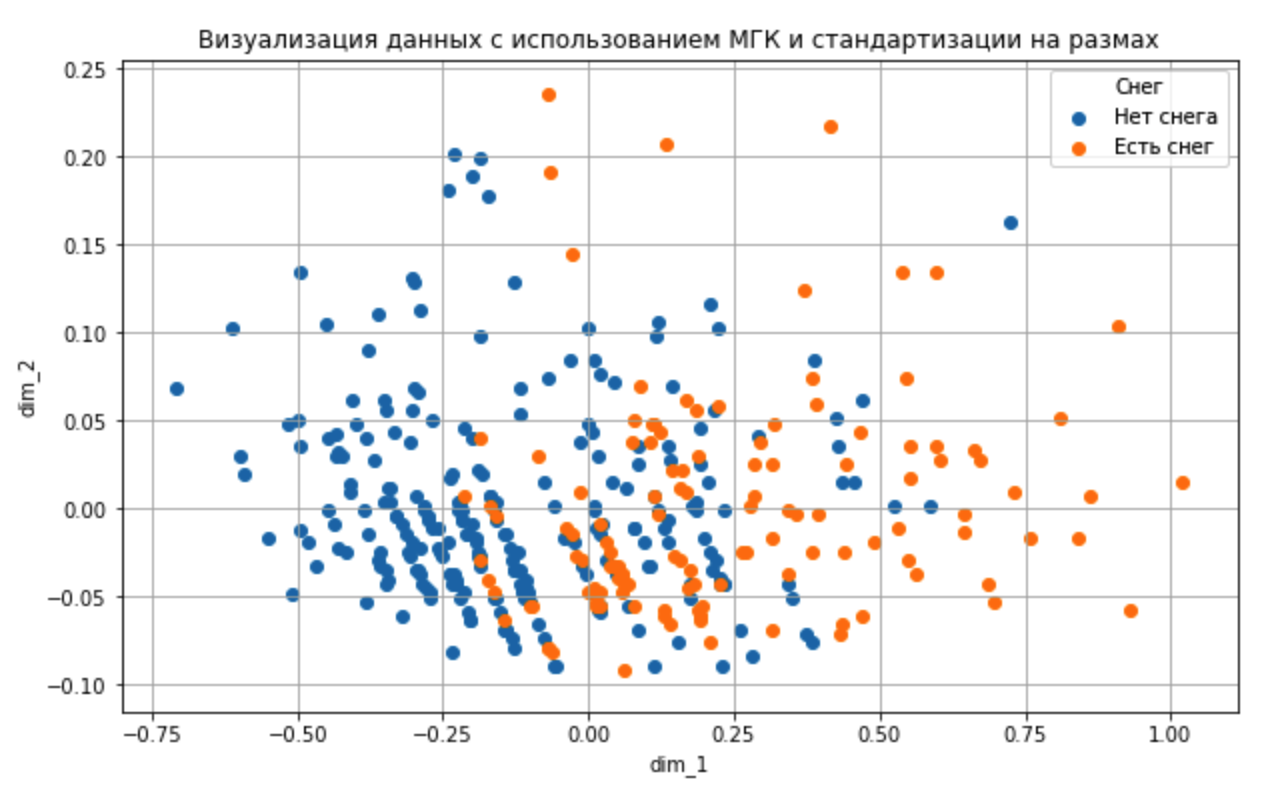


Иллюстрация 13 - Визуализация данных с использованием МГК и стандартизации на размах

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

1. **from** sklearn.decomposition **import** PCA
3. pca\_z = PCA(n\_components = 2)
4. XPCAreduced = pca\_z.fit\_transform(z\_scoring[['avg\_temp', 'min\_temp', 'max\_temp']].to\_numpy())
5. z\_s = pd.DataFrame(XPCAreduced, columns = ['col1', 'col2'])
6. z\_s['snow\_depth'] = df['snow\_depth'].to\_numpy()
8. fig, graph = plt.subplots(figsize=(10, 6))
9. plt.scatter(
10. z\_s[z\_s['snow\_depth'] == 0]['col1'],
11. z\_s[z\_s['snow\_depth'] == 0]['col2'],
12. label='Нет снега',
13. cmap='rainbow'
14. )
15. plt.scatter(
16. z\_s[z\_s['snow\_depth'] > 0]['col1'],
17. z\_s[z\_s['snow\_depth'] > 0]['col2'],
18. label='Снег',
19. cmap='rainbow'
20. )
22. plt.title('Визуализация данных с использованием МГК и z-scoring')
23. plt.xlabel('dim\_1')
24. plt.ylabel('dim\_2')
25. plt.grid()
26. plt.legend(title='Снег')

Изображение выглядит как небо

Автоматически созданное описание

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:

1. X = np.transpose(df[['avg\_temp', 'min\_temp', 'max\_temp']].to\_numpy())
2. Xcentered = (X[0] - X[0].mean(), X[1] - X[1].mean(), X[2] - X[2].mean())
3. m = (X[0].mean(), X[1].mean(), X[2].mean())
4. covmat = np.cov(Xcentered)
5. \_, vecs = np.linalg.eig(covmat)
6. v = [vecs[:,0], vecs[:,2]]
7. Xnew = pd.DataFrame(np.transpose(np.dot(v,Xcentered)), columns = ['col1', 'col2'])
8. Xnew['snow\_depth'] = df['snow\_depth'].to\_numpy()
10. fig, graph = plt.subplots(figsize=(10, 6))
11. plt.scatter(
12. Xnew[Xnew['snow\_depth'] == 0]['col1'],
13. Xnew[Xnew['snow\_depth'] == 0]['col2'],
14. label='Нет снега',
15. cmap='rainbow'
16. )
17. plt.scatter(
18. Xnew[Xnew['snow\_depth'] > 0]['col1'],
19. Xnew[Xnew['snow\_depth'] > 0]['col2'],
20. label='Есть снег',
21. cmap='rainbow'
22. )
24. plt.title('Визуализация данных с использованием традиционного МГК, реализованного вручную')
25. plt.legend(title='Снег')
26. plt.xlabel('dim\_1')
27. plt.ylabel('dim\_2')
28. plt.grid()

Изображение выглядит как небо, текст

Автоматически созданное описание

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:

1. **print** ("Первая главная компонента: ", v[0], "\nВторая главная компонента: ", v[1])
3. >>>Первая главная компонента:  [0.57942185 0.60603756 0.54496678]
4. >>>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
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])

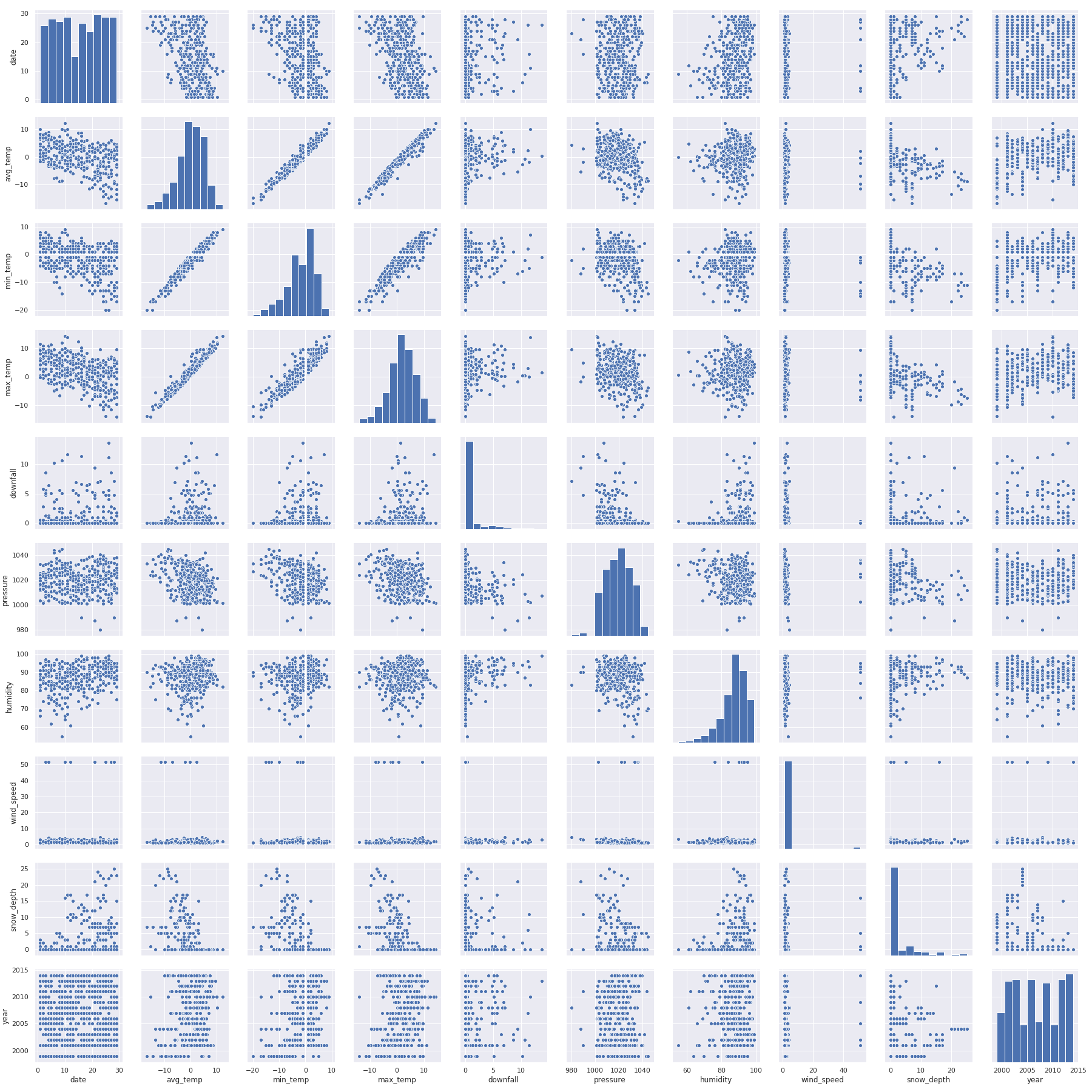


Иллюстрация 16 - Взаимосвязи между всеми признаками в наших данных

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:

1. fig, graph = plt.subplots(figsize=(13, 9))
2. **for** i **in** range(1, 31):
3. plt.scatter(
4. df[df.date == i].date,
5. df[df.date == i].avg\_temp,
6. label=f'{i} ноября',
7. cmap='rainbow'
8. )
9. plt.legend(loc='center left', bbox\_to\_anchor=(1, 0.5))

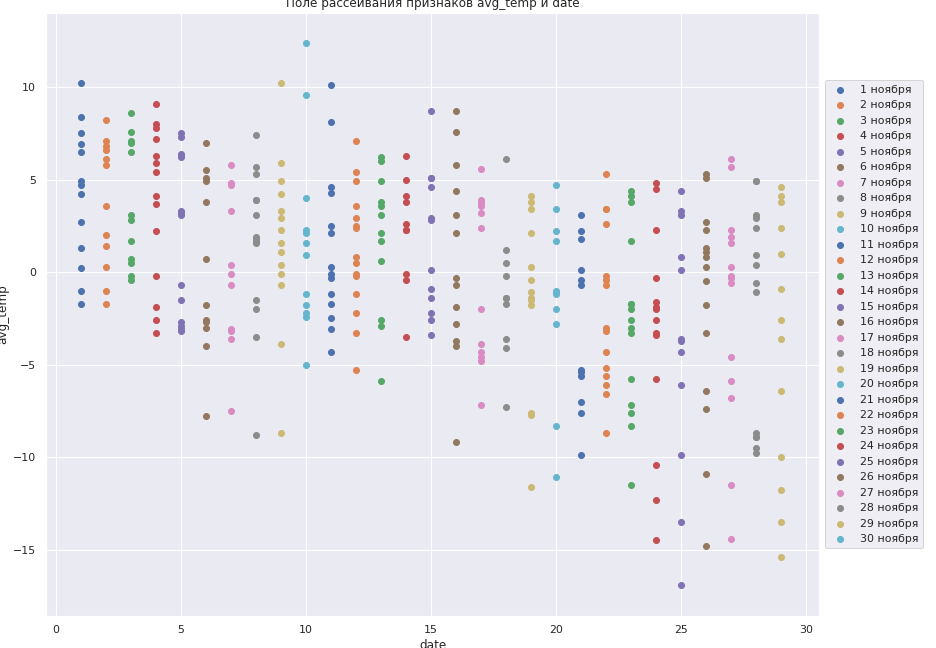


Иллюстрация 17 - Поле рассеивания признаков avg\_temp и date

Обучим линейную регрессию:

* date - признак
* avg\_temp - целевое значение

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

1. **from** sklearn.linear\_model **import** LinearRegression
3. model = LinearRegression()
4. model.fit(np.array(df['date'])[:,np.newaxis], np.array(df['avg\_temp'])[:,np.newaxis])
6. **print**(f'Уравнение линейной регрессии имеет вид:\ty = {model.coef\_.item()} \* x + {model.intercept\_.item()}')
8. >>>Уравнение линейной регрессии имеет вид: y = -0.2550634686459896 \* x + 3.9509900360879273

Визуализируем результат:

1. fig, graph = plt.subplots(figsize=(13, 9))
2. **for** i **in** range(1, 31):
3. plt.scatter(
4. df[df.date == i].date,
5. df[df.date == i].avg\_temp,
6. label=f'{i} ноября',
7. cmap='rainbow'
8. )
10. x = np.linspace(0,30)
11. y = model.coef\_.item()\*x + model.intercept\_.item()
12. plt.plot(x, y, label='Уравнение линейной регрессии')
13. plt.legend(loc='center left', bbox\_to\_anchor=(1, 0.5))

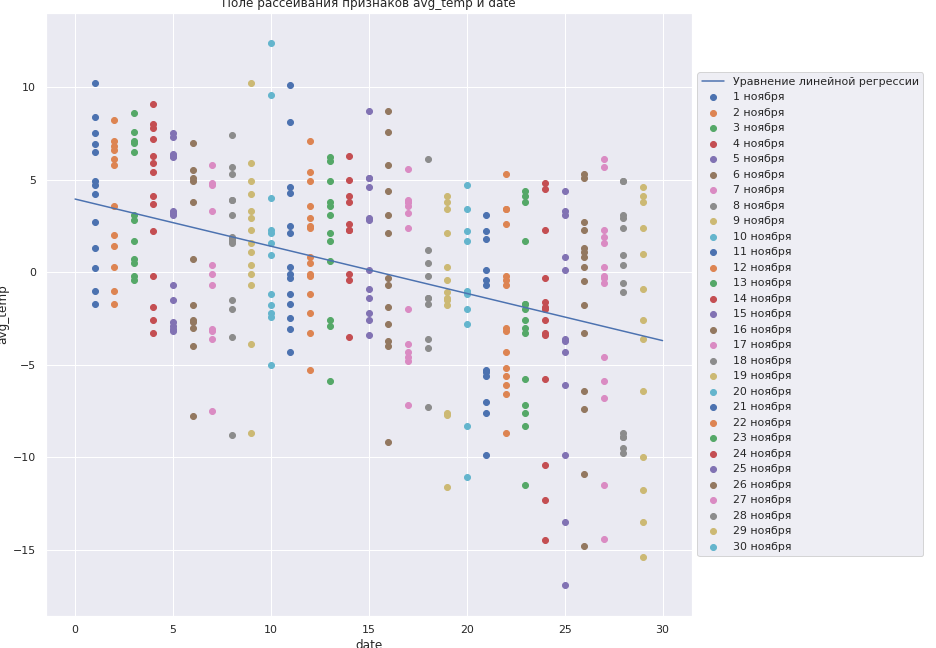


Иллюстрация 18 - Поле рассеивания признаков avg\_temp и date, уравнение линейной регрессии

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 R2) – функцией r2\_score[14] библиотеки SKLearn[7].

1. **from** scipy.stats **import** pearsonr
2. **from** sklearn.metrics **import** r2\_score
4. corr = pearsonr(model.coef\_.item()\*df['date'] + model.intercept\_.item(), df['avg\_temp'])[0]
5. r2 = r2\_score(df['avg\_temp'], model.coef\_.item()\*df['date'] + model.intercept\_.item())
7. **print**(f'Значение коэффициента корреляции Пирсона:\t{corr}')
8. **print**(f'Значение коэффициента детерминации:\t{r2}')
10. >>>Значение коэффициента корреляции Пирсона:   0.4204443991275562
11. >>>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.

1. **print**(f'''''
2. 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{model.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. ''')
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:   1.9104822869200104˙
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.intercept\_.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()}%')
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. <https://climate-energy.ru/weather/archive_weather_276120.php>
2. Ссылка на CSV (возможно, github)
3. <https://colab.research.google.com>
4. <https://pandas.pydata.org>
5. <https://pandas.pydata.org>
6. <https://matplotlib.org>
7. <https://scikit-learn.org/stable/>
8. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
9. <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>
10. <https://seaborn.pydata.org/generated/seaborn.pairplot.html>
11. <https://seaborn.pydata.org/index.html>
12. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html>
13. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html>
14. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2\_score.html