GeekBrains

Лекция 5. Google Colab(Jupyter). Знакомство с аналитикой.

Курс

Оглавление

Введение

Знакомство с аналитикой. Мы будем пользоваться таким инструментом как Google colab

На лекции мы познакомимся с инструментом для работы с табличными данными(**pandas**) и способами визуализации данных с помощью библиотек **matplotlib** и **seaborn**. Прежде, чем приступать непосредственно к машинному обучению, важно произвести **EDA**(Exploratory Data Analysis) - Разведочный анализ данных.

Он состоит в анализе основных свойств данных, нахождения в них общих закономерностей, распределений и аномалий, построение начальных моделей, зачастую с использованием инструментов визуализации.

Понятие введено математиком **Джоном Тьюки**, который сформулировал цели такого анализа следующим образом:

1. Максимальное «проникновение» в данные
2. Выявление основных структур
3. Выбор наиболее важных переменных
4. Обнаружение отклонений и аномалий
5. Проверка основных гипотез

Термины, используемые в лекции

Перцентиль - это показатель, используемый в статистике, показывающий значение, ниже которого падает определенный процент наблюдений в группе наблюдений

Scatterplot (Точечный график) - Математическая диаграмма, изображающая значения двух переменных в виде точек на декартовой плоскости.

Медиана набора чисел — число, которое находится в середине этого набора, если его упорядочить по возрастанию, то есть такое число, что половина из элементов набора не меньше него, а другая половина не больше.

Базовые функции для работы с данными

Библиотека **pandas** может читать многие форматы, включая: **.csv, .xslx, .xls, .txt, sql** и многие другие. Полный список по ссылке

Чтобы подключить библиотеку к Вашей программе необходимо написать следующее:

```
import pandas as pd
```

Напоминание: as(alias) - псевдоним. Мы можем сократить название все библиотеки до 2-х букв.

Прочтем файл .csv(он находится в Google Colab в папке **sample_data**) с помощью библиотеки **pandas**

```
df = pd.read_csv('sample_data/california_housing_train.csv')
```

Для того чтобы прочитать первые n строк таблицы, необходимо воспользоваться следующей функцией:

```
DataFrame.head(n=5)
```

Где DataFrame - это таблица с данными, которая предварительно была открыта. Мы ее открыли и записали в переменную **df.** Необязательно указывать n=5, вместо 5 мы можем указать любое число(число не должно превосходить количество строк в таблице). Если Вы ничего не укажете в круглых скобках, то ошибка не вылезет, по умолчанию будут выведены первые 5 строк таблицы.

Пример:

```
df.head()
```

Результат:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

Как мы знаем, в нашем мире почти все симметрично, есть отрицательные числа, а есть положительные и тд. Значит, если есть функция, которая показывает первые 5 строк таблицы, то и есть функция, которая показывает последние 5 строк таблицы. Да, действительно, это так. Давайте с ней познакомимся

Пример:

```
df.tail()
```

Результат:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
16995	-124.26	40.58	52.0	2217.0	394.0	907.0	369.0	2.3571	111400.0
16996	-124.27	40.69	36.0	2349.0	528.0	1194.0	465.0	2.5179	79000.0
16997	-124.30	41.84	17.0	2677.0	531.0	1244.0	456.0	3.0313	103600.0
16998	-124.30	41.80	19.0	2672.0	552.0	1298.0	478.0	1.9797	85800.0
16999	-124.35	40.54	52.0	1820.0	300.0	806.0	270.0	3.0147	94600.0

Данная функция работает аналогично с head(). Необязательно выводить последние 5 строчек, можно указать сколько угодно.

Иногда заранее неизвестно сколько строк и столбцов находится внутри таблицы, чтобы это сделать необходимо воспользоваться специальной функцией.

Пример:

```
df.shape
```

Результат:

```
(17000, 9)
```

Функция shape возвращает размеры таблицы: кортеж из 2 значений, 1 - количество строк, 2 - количество столбцов.

Согласитесь, что все не раз делали заказ на каком-нибудь маркетплейсе. И когда мы заполняли поле **“Email”**, то могли его пропустить, потому что указанно, что оно необязательное и не хотели видеть лишнего спама. Вы когда-нибудь задумывались, как в этом случае эти данные будут выглядеть внутри базы данных(таблице)? Пропуск? Пустая ячейка? Нет. Когда нужно указать, что в данной ячейки таблицы ничего нет указывается значение **null**.

Чтобы обнаружить пустые значения в таблице данных необходимо воспользоваться функцией **.isnull()**.

Пример:

```
df.isnull()
```

Результат:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
16995	False	False	False	False	False	False	False	False	False
16996	False	False	False	False	False	False	False	False	False
16997	False	False	False	False	False	False	False	False	False
16998	False	False	False	False	False	False	False	False	False
16999	False	False	False	False	False	False	False	False	False

Функция привела нашу таблицу к следующему виду **True-False,** где **True** - это пустая ячейка, **False** - это заполненная ячейка. Но это неудобно, то есть нам надо просматривать 17 000 * 9 = 153 000 ячеек. Вау... Это займет слишком много времени. Однако, мы можем воспользоваться еще одной функцией .**sum()**. Данная функция выведет количество null-значений в каждой ячейке по столбцам.

Пример:

```
df.isnull().sum()
```

Результат:

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
```

Можно сделать следующий вывод: пустые значения в нашей таблицы отсутствуют.

Еще при работе с C#, мы поняли, что у каждой переменной есть свой тип данных. Также и здесь, у каждого столбца есть свой тип данных, чтобы это узнать, нужно применить функцию **.dtypes**.

Пример:

```
df.dtypes
```

Результат:

```
longitude             float64
latitude              float64
housing_median_age    float64
total_rooms           float64
total_bedrooms        float64
population            float64
households            float64
median_income         float64
median_house_value    float64
```

Делаем вывод: у всей таблицы данных один тип **float(дробное число)**

Чтобы узнать название всех столбцов в таблице, воспользуйтесь функцией **.columns.**

Пример:

```
df.columns
```

Результат:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

Выборка данных

💡 **Медиана набора чисел** — число, которое находится в середине этого набора, если его упорядочить по возрастанию, то есть такое число, что половина из элементов набора не меньше него, а другая половина не больше.

Если Вы хотите вывести 1 столбец на экран, то можно указать следующее выражение, которое позволит это сделать.

Пример:

```
df['latitude']
```

Результат:

```
0        34.19
1        34.40
2        33.69
3        33.64
4        33.57
        ...
16995    40.58
16996    40.69
16997    41.84
16998    41.80
16999    40.54
Name: latitude, Length: 17000, dtype: float64
```

Что мы будем делать, если нам потребуется вывести на экран сразу несколько столбцов? Не очень удобно будет это прописывать вот таким образом:

```
print(df['latitude'])

print(df['population'])
```

Конечно есть решение данного вопроса

Пример:

```
df[['latitude', 'population']]
```

Результат:

	latitude	population
0	34.19	1015.0
1	34.40	1129.0
2	33.69	333.0
3	33.64	515.0
4	33.57	624.0
...	...	...
16995	40.58	907.0
16996	40.69	1194.0
16997	41.84	1244.0
16998	41.80	1298.0
16999	40.54	806.0

Задание: Необходимо вывести столбец **total_rooms**, у которого медианный возраст здания(**housing_median_age**) меньше **20**.

Для того чтобы решить это задание, давайте познакомимся с синтаксисом выборки данных. На самом деле, это ничем не отличается от операторов ветвления.

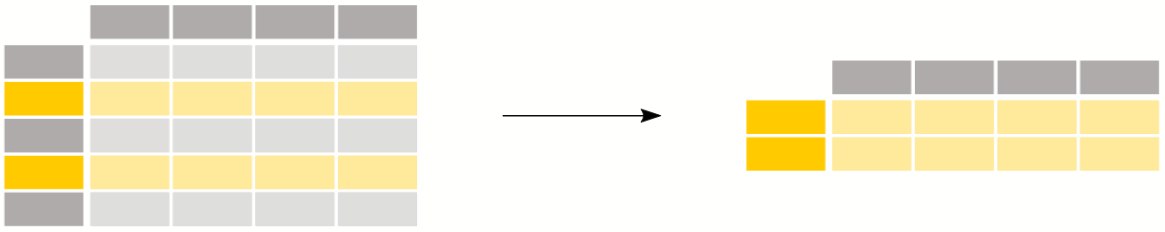
Решение:

```
df[df['housing_median_age'] < 20]
```

Результат:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
10	-114.60	33.62	16.0	3741.0	801.0	2434.0	824.0	2.6797	86500.0
...	...	...	...	...	...	...	...	...	...
16983	-124.19	41.78	15.0	3140.0	714.0	1645.0	640.0	1.6654	74600.0
16987	-124.21	41.77	17.0	3461.0	722.0	1947.0	647.0	2.5795	68400.0
16991	-124.23	41.75	11.0	3159.0	616.0	1343.0	479.0	2.4805	73200.0
16997	-124.30	41.84	17.0	2677.0	531.0	1244.0	456.0	3.0313	103600.0
16998	-124.30	41.80	19.0	2672.0	552.0	1298.0	478.0	1.9797	85800.0

4826 rows × 9 columns



Если Вам нужно поставить другое условие, то аналогично.

Мы помним с C#, что иногда приходится проверять несколько условий сразу. Чтобы проверить несколько условий внутри Google Colab, указывается так:

```
df[(df['housing_median_age'] > 20) & (df['total_rooms'] > 2000)]
```

& - выполнение одновременно **всех** условий.

| - выполнение **хотя бы одного** из условия.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
6	-114.58	33.61	25.0	2907.0	680.0	1841.0	633.0	2.6768	82400.0
8	-114.59	33.61	34.0	4789.0	1175.0	3134.0	1056.0	2.1782	58400.0
13	-114.61	34.83	31.0	2478.0	464.0	1346.0	479.0	3.2120	70400.0
42	-115.49	32.67	25.0	2322.0	573.0	2185.0	602.0	1.3750	70100.0
45	-115.50	32.67	35.0	2159.0	492.0	1694.0	475.0	2.1776	75500.0
...	...	...	...	...	...	...	...	...	...
16986	-124.19	40.73	21.0	5694.0	1056.0	2907.0	972.0	3.5363	90100.0
16990	-124.22	41.73	28.0	3003.0	699.0	1530.0	653.0	1.7038	78300.0
16993	-124.23	40.54	52.0	2694.0	453.0	1152.0	435.0	3.0806	106700.0
16995	-124.26	40.58	52.0	2217.0	394.0	907.0	369.0	2.3571	111400.0
16996	-124.27	40.69	36.0	2349.0	528.0	1194.0	465.0	2.5179	79000.0

5624 rows × 9 columns

Первую часть задания мы успешно выполняли! Только загвоздка... Нам не нужна вся таблица, а лишь один столбец, как это сделать?

Решение:

```
df[df['housing_median_age'] < 20, 'total_rooms']

# или (если необходимо вывести 2 и более столбцов

df[df['housing_median_age'] < 20, ['total_bedrooms', 'total_rooms']]
```

Результат:

```
17         44.0
19         97.0
113        96.0
116       208.0
120       186.0
          ...
16643     255.0
16733     411.0
16743      89.0
16801      98.0
16851     133.0
Name: total_rooms, Length: 178, dtype: float64
```

Простая статистика

Pandas позволяет получить основные простые данные для описательной статистики. Такие как минимальное значение в столбце, максимальное значение, сумма всех значений, среднее значение

Максимальное значение:

```
print(df['population'].max())
```

Результат:

```
35682.0
```

Минимальное значение:

```
print(df['population'].min())
```

Результат:

```
3.0
```

Среднее значение:

```
print(df['population'].mean())
```

Результат:

```
1429.5739411764705
```

Сумма:

```
print(df['population'].sum())
```

Результат:

```
24302757.0
```

Эту же статистику можно рассчитывать сразу для нескольких столбцов

Медианное значение:

```
df[['population', 'total_rooms']].median()
```

Результат:

```
population      1155.0
total_rooms     2106.0
dtype: float64
```

💡 **Перцентиль** - это показатель, используемый в статистике, показывающий значение, ниже которого падает определенный процент наблюдений в группе наблюдений

Получить общую картину можно простой командой **describe**

```
df.describe()
```

Результат:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	3000.000000	3000.00000	3000.000000	3000.000000	3000.000000	3000.000000	3000.00000	3000.000000	3000.00000
mean	-119.589200	35.63539	28.845333	2599.578667	529.950667	1402.798667	489.91200	3.807272	205846.27500
std	1.994936	2.12967	12.555396	2155.593332	415.654368	1030.543012	365.42271	1.854512	113119.68747
min	-124.180000	32.56000	1.000000	6.000000	2.000000	5.000000	2.00000	0.499900	22500.00000
25%	-121.810000	33.93000	18.000000	1401.000000	291.000000	780.000000	273.00000	2.544000	121200.00000
50%	-118.485000	34.27000	29.000000	2106.000000	437.000000	1155.000000	409.50000	3.487150	177650.00000
75%	-118.020000	37.69000	37.000000	3129.000000	636.000000	1742.750000	597.25000	4.656475	263975.00000
max	-114.490000	41.92000	52.000000	30450.000000	5419.000000	11935.000000	4930.00000	15.000100	500001.00000

count - Общее кол-во не пустых строк

mean - среднее значение в столбце

std - стандартное отклонение от среднего значения

min - минимальное значение

max - максимальное значение

Числа **25%, 50%, 75%** - перцентили

Изображаем статистические отношения

Scatterplot (Точечный график)

Математическая диаграмма, изображающая значения двух переменных в виде точек на декартовой плоскости. Библиотека **seaborn** без труда принимает **pandas DataFrame**(таблицу). Чтобы изобразить отношения между двумя столбцами достаточно указать, какой столбец отобразить по оси **x**, а какой по оси **y**.

Для того чтобы начать работу с библиотекой seaborn, ее необходимо импортировать к себе в программу:

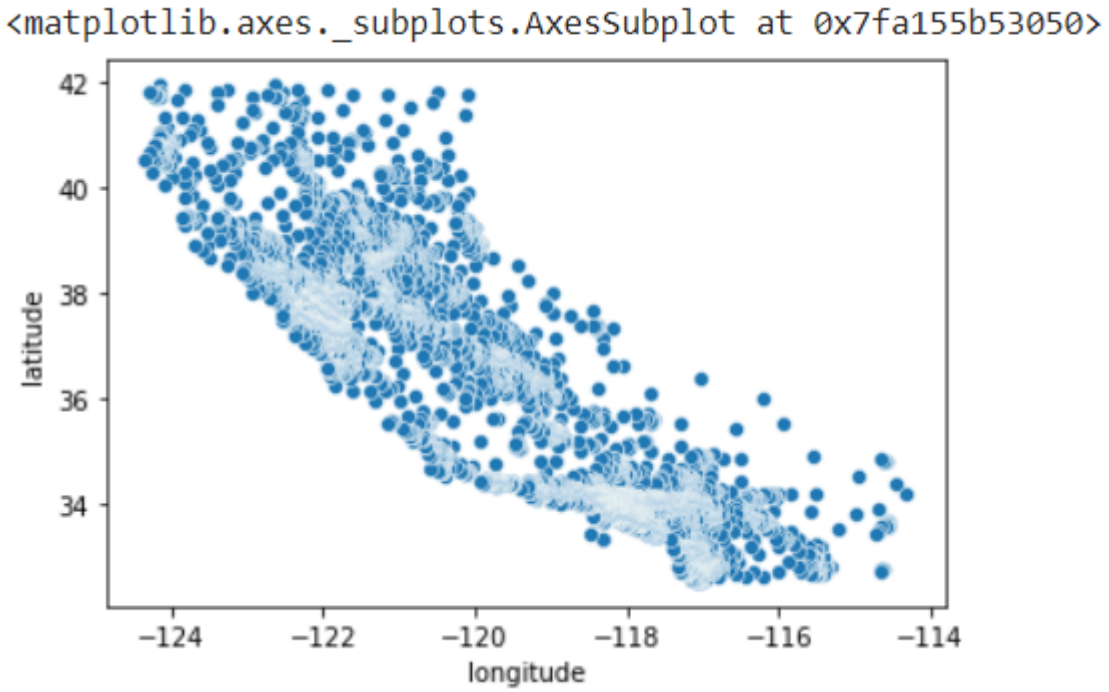
```
import seaborn as sns
```

Вернемся к нашей таблице. Можно заметить, что дома расположены в определенной "полосе" долготы и широты.

Изображение точек долготы по отношению к широте:

```
sns.scatterplot(data=df, x="longitude", y="latitude")
```

Результат:

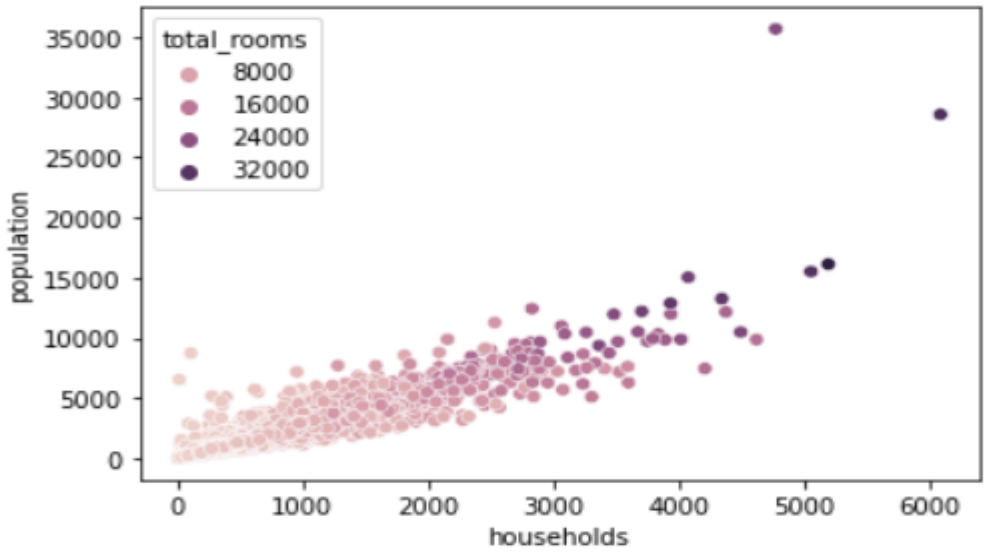


Помимо двумерных отношений, мы можем добавить "дополнительное измерение" с помощью цвета. В данном случае опять же достаточно очевидное отношение, чем выше кол-во семей, тем выше кол-во людей и соответственно комнат.

```
sns.scatterplot(data=df, x="households", y="population",  hue="total_rooms")
```

Результат:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa14df5bcd0>
```

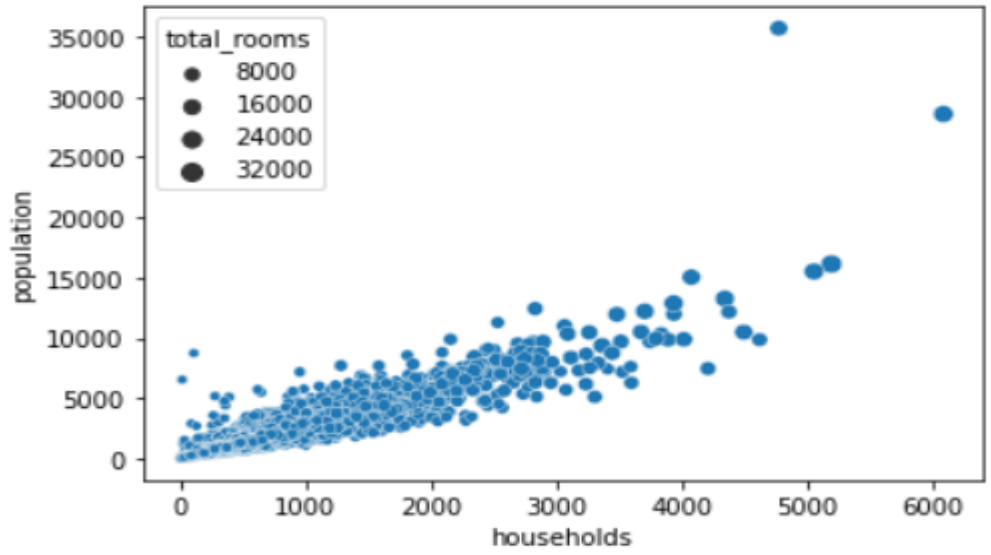


Помимо обозначения дополнительного измерения цветом мы можем использовать **size**.

```
sns.scatterplot(data=df, x="households", y="population",  hue="total_rooms")
```

Результат:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa14c1a3490>
```



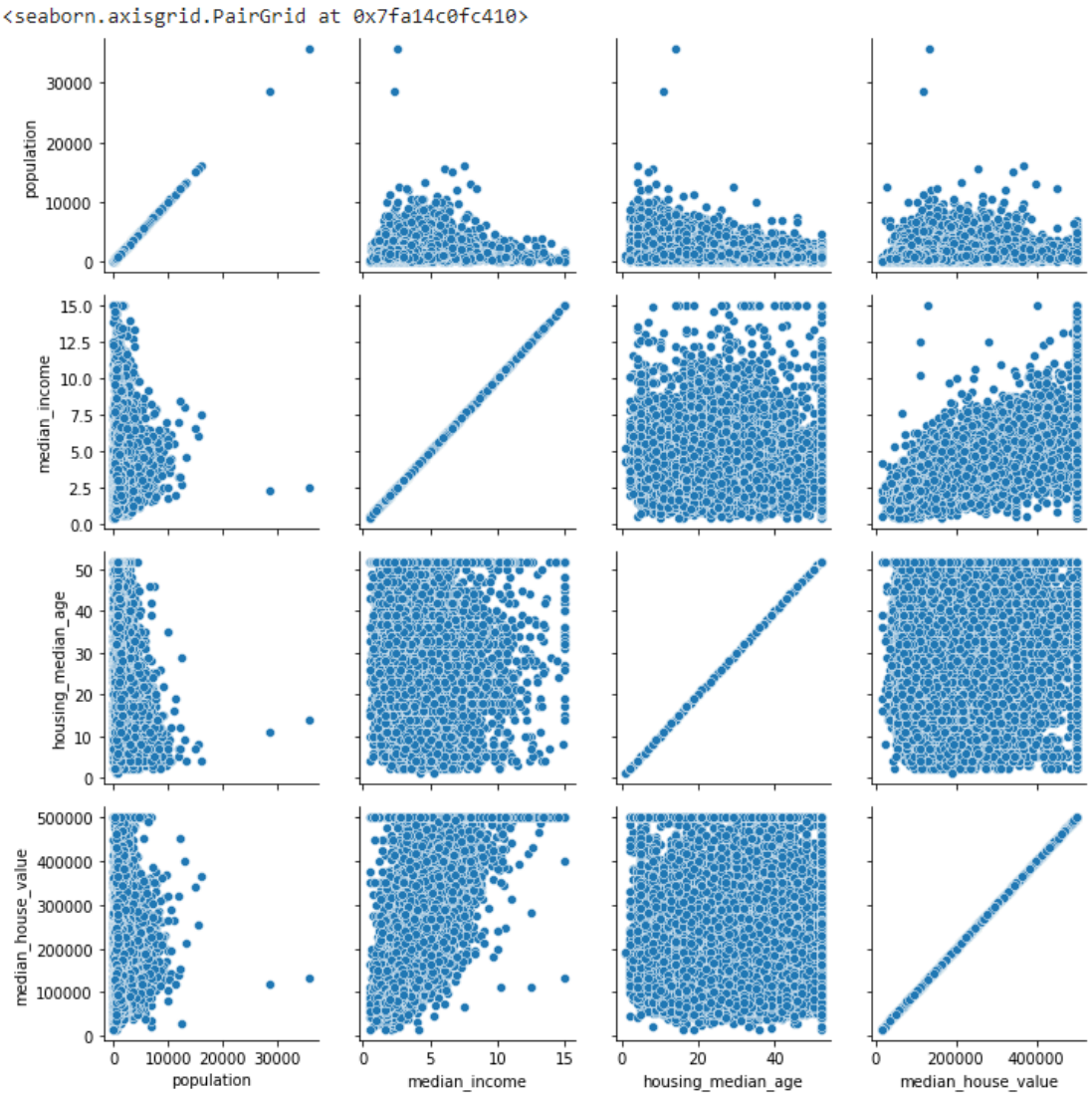
Мы можем визуализировать сразу несколько отношений, используя класс **PairGrid** внутри **seaborn. PairGrid** принимает как аргумент pandas **DataFrame** и визуализирует все возможные отношения между ними, в соответствии с выбранным типом графика.

```
cols = ['population', 'median_income', 'housing_median_age', 'median_house_value']

g = sns.PairGrid(df[cols])

g.map(sns.scatterplot)
```

Результат:



Как Вы думаете, чем вызвана линейная зависимость по диагонали?

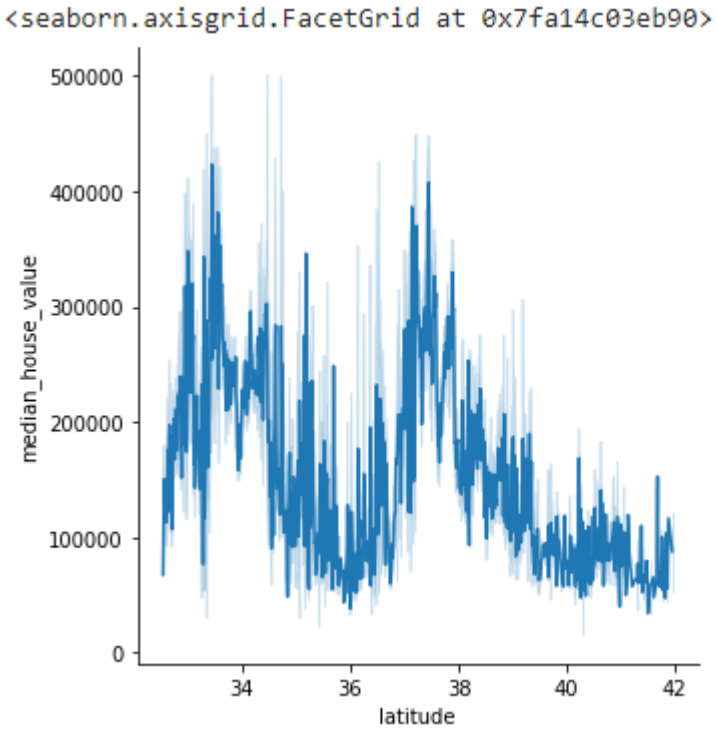
Линейные графики

Хорошо подойдут, если есть временная или какая-либо иная последовательность и значения, которые могут меняться в зависимости от нее. Для генерации линейных графиков в **seaborn** используется **relplot** функцию. Она также принимает **DataFrame**, **x**, **y** - столбцы.

Для визуализации выбирается тип **line**:

```
sns.relplot(x="latitude", y="median_house_value", kind="line", data=df)
```

Результат:



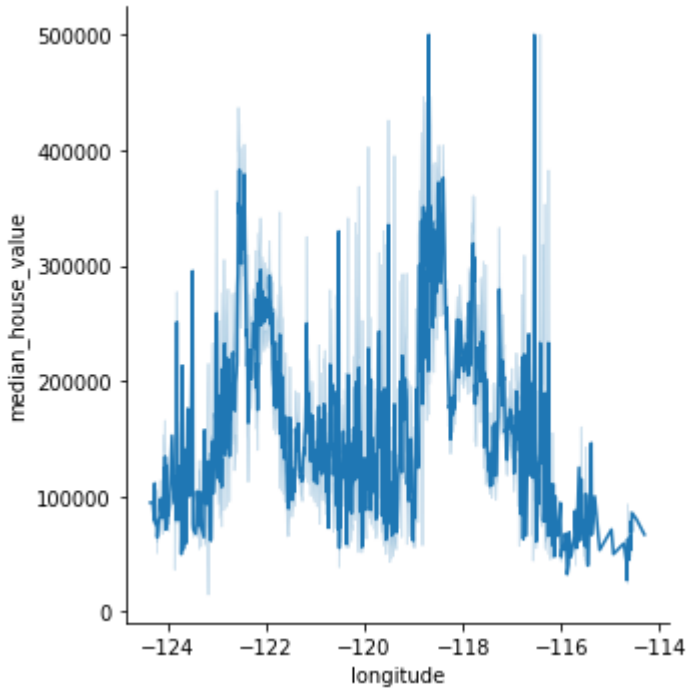
Можно видеть, что в определенных местах долготы цена за дома резко подскакивает.

Попробуем визуализировать longitude по отношения к median_house_value и поймем в чем же дело, почему цена так резко подскакивает.

```
sns.relplot(x = 'longitude', y = 'median_house_value', kind = 'line', data = df)
```

Результат:

```
<seaborn.axisgrid.FacetGrid at 0x7fa14c180b10>
```



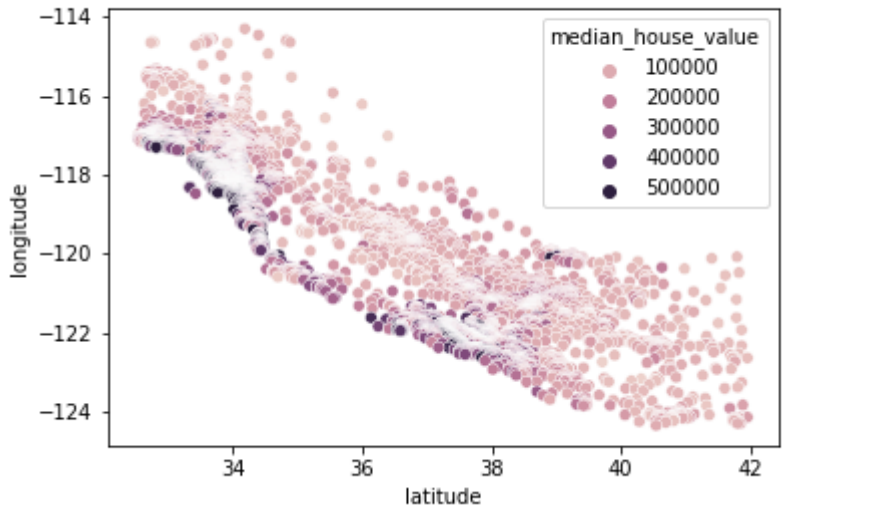
Можно видеть, что в определенных местах широты цена за дома также очень высока.

Используя точечный график можно визуализировать эти отношения с большей четкостью. Скорее всего резкий рост цен связан с близостью к ценному объекту, повышающему качество жизни, скорее всего побережью океана или реки.

```
sns.scatterplot(data=df, x="latitude", y="longitude",  hue="median_house_value")
```

Результат:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa149121fd0>
```



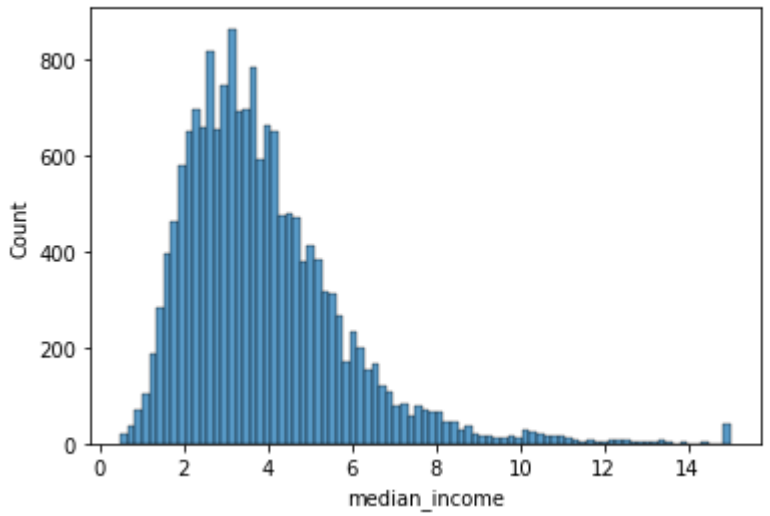
Гистограмма

Способ представления табличных данных в графическом виде — в виде столбчатой диаграммы. По оси **x** обычно указывают значение, а по оси **y** - встречаемость(кол-во таких значений в выборке)

```
sns.histplot(data=df, x="median_income")
```

Результат:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa14915eb10>
```



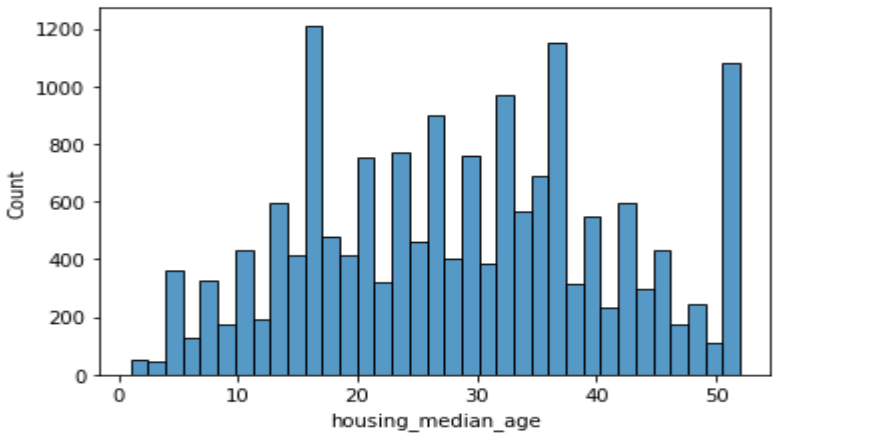
Можно видеть что у большинства семей доход находится между значениями 2 и 6. И только очень небольшое количество людей обладают доходом > 10.

Изобразим гистограмму по **housing_median_age.**

```
sns.histplot(data = df, x = 'housing_median_age')
```

Результат:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa148fc6c50>
```



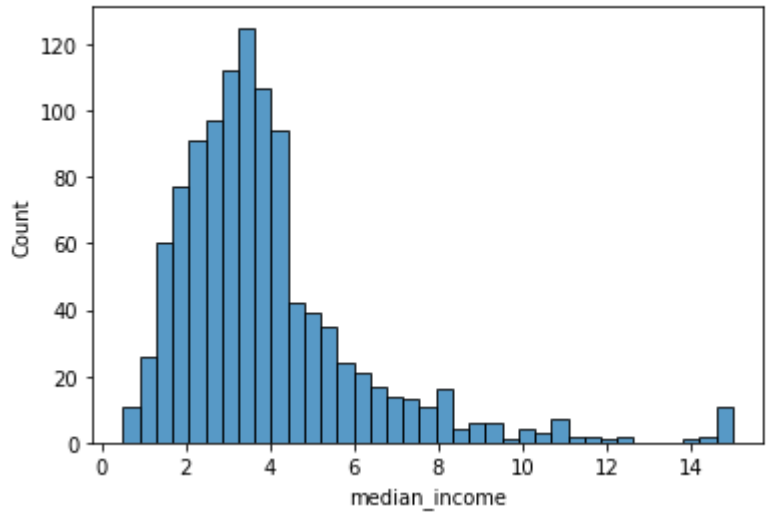
Распределение по возрасту более равномерное. Большую часть жителей составляют люди в возрасте от 20 до 40 лет. Но и молодежи не мало. Также очень много пожилых людей > 50 лет медианный возраст.

Давайте посмотрим медианный доход у пожилых жителей.

```
sns.histplot(data=df[df['housing_median_age']>50], x="median_income")
```

Результат:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa1490f3350>
```



Большого отличия от популяции в целом не наблюдается. Скорее всего это местные жители.

Давайте разобьем возрастные группы на 3 категории те кто моложе 20 лет, от 20 до 50 и от 50, чтобы посмотреть влияет ли это на доход.

```
df.loc[df['housing_median_age'] <= 20, 'age_group'] = 'Молодые'

df.loc[(df['housing_median_age'] > 20) & (df['housing_median_age'] <= 50),
'age_group'] = 'Ср. возраст'

df.loc[df['housing_median_age'] > 50, 'age_group'] = 'Пожилые'
```

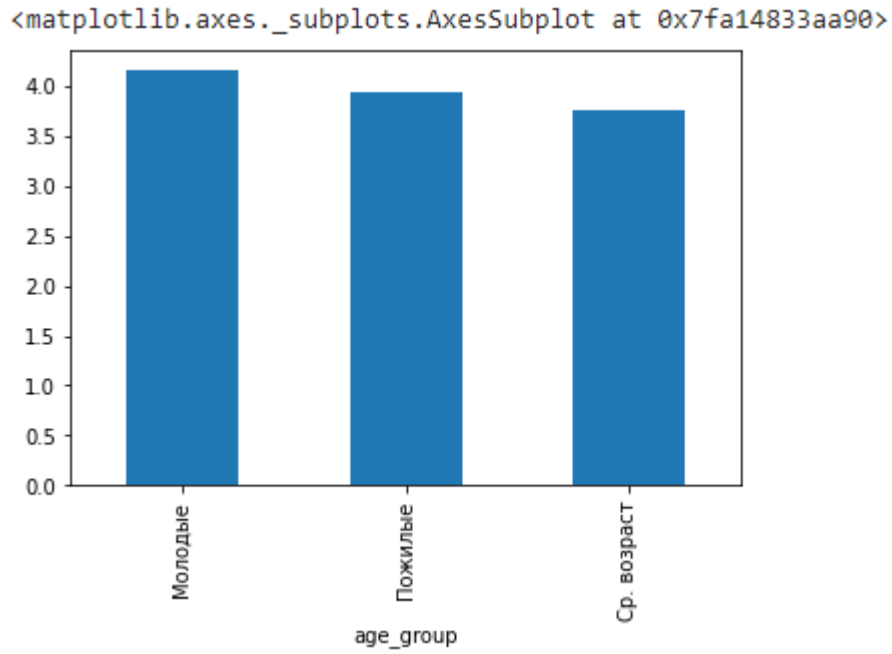
Что в этом случае происходит внутри таблицы? Добавился новый столбец **age_group**, в котором будет указана соответствующая категория.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	age_group
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	Молодые

Применим **group_by**, чтобы получить среднее значение.

```
df.groupby('age_group')['median_income'].mean().plot(kind='bar')
```

Результат:



Молодые оказываются самой богатой группой населения. Но отличие в доходе не значительное.

Seaborn так же позволяет нам смотреть распределение по многим параметрам. Давайте поделим группы по доходам на 2. Те у кого медианный доход выше 6 и те у кого меньше. Изобразим дополнительное измерение с помощью оттенка в виде возрастных групп и групп по доходам.

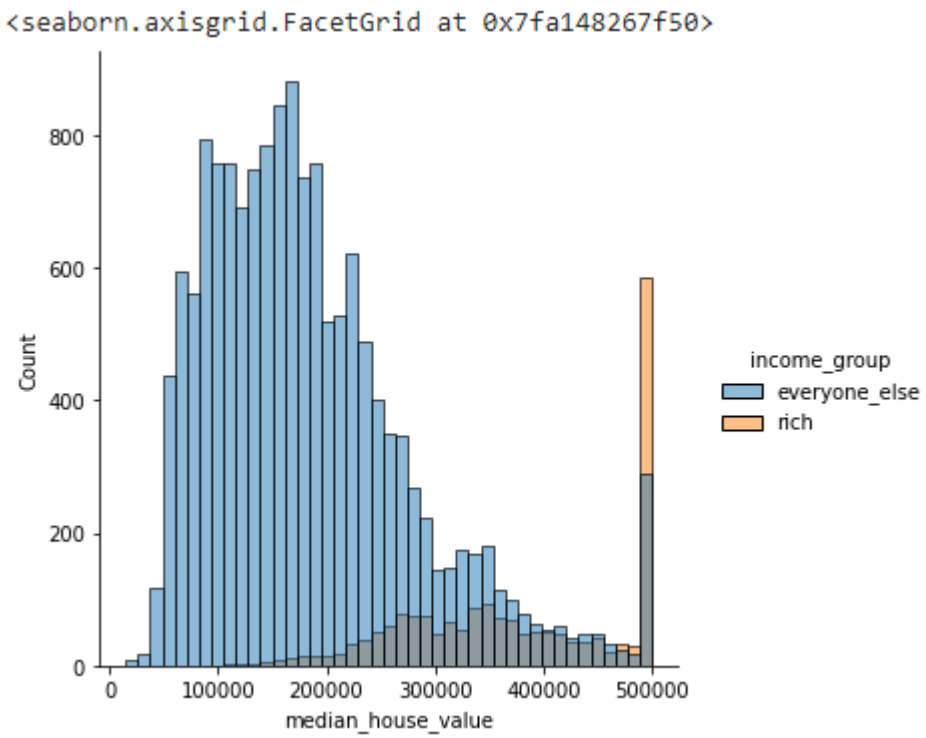
```
df.loc[df['median_income'] > 6, 'income_group'] = 'rich'

df.loc[df['median_income'] < 6, 'income_group'] = 'everyone_else'
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	age_group	income_group
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0	Молодые	everyone_else

```
sns.displot(df, x="median_house_value", hue="income_group")
```

Результат:



Итоги:

Анализ данных должен предоставлять информацию и инсайт, которые не видны невооруженным взглядом. В этом и есть красота аналитики. В данном случае можно сделать следующий выводы. Стоимость домов напрямую зависит от их расположения, в определенной полосе(скорее всего побережье) цена на дома высокая. Чем выше доход, тем больше шанс, что человек проживает в богатом районе. Распределение по возрастам примерно одинаковое во всех группах доходов. Ну и очевидно чем больше людей, тем больше семей, и соответственно комнат и спален.