Примеры использования библиотек Python

Цель занятия

После освоения темы:

- вы узнаете некоторые типичные приемы по анализу данных;
- сможете выбрать библиотеки Python для решения задачи анализа данных;
- сможете выполнять анализ данных из одного и нескольких источников с использованием языка Python;
- сможете строить и анализировать матрицу корреляции на Python;
- сможете визуализировать данные с помощью различных видов графиков, строить интерактивные графики;
- сможете использовать функциональные возможности нескольких библиотек в одном проекте.

План занятия

Конспект занятия

1. Пример анализа данных с помощью библиотеки `Pandas`

Сформулируем задачу, с которой мы сталкиваемся как аналитики. Пусть мы работаем в некотором стриминговом сервисе, и нас интересует, какой контент из аниме стоит добавить в сервис.

Для этого изначально нужно получить некоторые данные по оценкам аниме. Мы нашли некие сторонние данные с сайта myanimelist.net, в котором хранятся оценки 76 тыс. пользователей. На основе этих данных мы должны провести ряд выкладок, чтобы понять, какие аниме стоит добавить. То есть конечная цель исследования — выделить лучшие аниме, чтобы увеличить количество просмотров на нашем сервисе. Результаты исследования мы планируем передать заказчику и команде, которая будет договариваться о покупке контента.

Импортируем необходимые нам библиотеки `Pandas` и `NumPy`, подгрузим csv-файлы (`anime.csv` и `rating.csv`)[1]:

```
import pandas as pd
import numpy as np
anime = pd.read_csv('anime.csv', sep=',')
rating = pd.read_csv('rating.csv', sep=',')
```

Посмотрим, какие данные хранятся в DataFrame в объекте `anime`. Выведем на экран первые пять строк:

```
anime.head()
```

Результат выполнения:

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

[1] В этом примере мы используем датасет Anime Recommendations Database

Как мы видим, DataFrame содержит:

- `anime_id` — некий идентификатор аниме;
- `name` — название;
- `genre` — жанр;
- `type` — тип (ТВ-шоу, фильм);
- `episodes` — количество эпизодов;
- `rating` — средний рейтинг;
- `members` — количество человек, подписанных на аниме.

Посмотрим на размерность:

```
anime.shape
```

Результат выполнения: `(12294, 7)`. То есть DataFrame имеет 12294 строки и 7 колонок.

Посмотрим более подробную информацию о данных с помощью метода `info()`:

```
anime.info()
```

Результат выполнения:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12294 entries, 0 to 12293
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   anime_id  12294 non-null  int64  
 1   name      12294 non-null  object 
 2   genre     12232 non-null  object 
 3   type      12269 non-null  object 
 4   episodes  12294 non-null  object 
 5   rating    12064 non-null  float64
 6   members   12294 non-null  int64  
dtypes: float64(1), int64(2), object(4)
memory usage: 672.5+ KB
```

Проделаем аналогичные операции для `rating`. Получаем первые 5 строк:

```
rating.head()
```

Результат выполнения:

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

DataFrame содержит:

- `user_id` — идентификатор пользователя;
- `anime_id` — идентификатор аниме, который пользователь оценил;
- `rating` — оценка, поставленная пользователем.

Оценка имеет значения от «1» до «10». В случае если пользователь посмотрел аниме, но не оценил его, в таблицу будет проставлено «–1».

Получаем размерность:

```
rating.shape
```

Результат выполнения: `(7813737, 3)`. То есть DataFrame содержит 7813737 строк и 3 столбца.

Получаем подробную информацию о DataFrame:

```
rating.info()
```

Результат выполнения:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7813737 entries, 0 to 7813736
Data columns (total 3 columns):
 #   Column    Dtype
---  ------    -----
 0   user_id   int64
 1   anime_id  int64
 2   rating    int64
dtypes: int64(3)
memory usage: 178.8 MB
```

Выведем сводную информацию по описательным статистикам из двух исследуемых DataFrame. Воспользуемся методом `describe()`:

```
anime.describe()
rating.describe()
```

Результат выполнения для объекта `anime`:

	anime_id	rating	members
count	12294.000000	12064.000000	1.229400e+04
mean	14058.221653	6.473902	1.807134e+04
std	11455.294701	1.026746	5.482068e+04
min	1.000000	1.670000	5.000000e+00
25%	3484.250000	5.880000	2.250000e+02
50%	10260.500000	6.570000	1.550000e+03
75%	24794.500000	7.180000	9.437000e+03
max	34527.000000	10.000000	1.013917e+06

Как мы видим, среднее значение рейтинга приблизительно равно 6,5; стандартное отклонение — около 1; минимальный рейтинг — 1,67.

Результат выполнения для объекта `rating`:

	user_id	anime_id	rating
count	7.813737e+06	7.813737e+06	7.813737e+06
mean	3.672796e+04	8.909072e+03	6.144030e+00
std	2.099795e+04	8.883950e+03	3.727800e+00
min	1.000000e+00	1.000000e+00	-1.000000e+00
25%	1.897400e+04	1.240000e+03	6.000000e+00
50%	3.679100e+04	6.213000e+03	7.000000e+00
75%	5.475700e+04	1.409300e+04	9.000000e+00
max	7.351600e+04	3.451900e+04	1.000000e+01

На первый взгляд никаких аномальных значений нет.

Проверим DataFrame `anime` на наличие пропусков:

```
anime.isnull().mean().sort_values(ascending=False)
```

Результат выполнения:

```
rating      0.018708
genre       0.005043
type        0.002034
anime_id    0.000000
name        0.000000
episodes    0.000000
members     0.000000
dtype: float64
```

Как видно, больше всего пропусков встречается в значении рейтинга, также имеются пропуски в колонках «жанр» и «тип».

Далее рассмотрим числовые признаки: минимум, максимум, среднее и медианное значения. Минимальное значение:

```
anime.min()
```

Результат выполнения:

```
<ipython-input-21-050890665a70>:1: FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the
reduction.

anime_id                1
name        &amp;quot;0&amp;quot;
episodes                1
rating               1.67
members                 5
dtype: object
```

Максимальное значение:

```
anime.max()
```

Результат выполнения:

```
<ipython-input-22-ff90c9aba999>:1: FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will
```

```
raise TypeError.  Select only valid columns before calling the
reduction.

anime_id      34527
name              ◯
episodes    Unknown
rating         10.0
members     1013917
dtype: object
```

Среднее значение:

```
anime.mean()
```

Результат выполнения:

```
<ipython-input-23-27334d35352e>:1: FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the
reduction.

anime_id    14058.221653
rating          6.473902
members     18071.338864
dtype: float64
```

Медианное значение:

```
anime.median()
```

Результат выполнения:

```
<ipython-input-24-db0a1e174c1a>:1: FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the
reduction.

anime_id    10260.50
rating          6.57
members      1550.00
dtype: float64
```

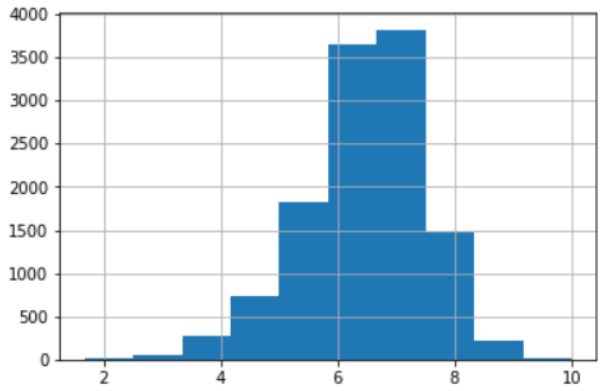
Посмотрев верхнеуровнево на данные (в том числе с помощью некоторых числовых признаков), мы не выявили каких-либо аномалий.

Воспользуемся гистограммой частот, чтобы посмотреть на выбросы. Если в большом объеме данных существует несколько выбросов с аномальными значениями, нам явно об этом стоит знать.

Построим гистограмму с помощью встроенных в `Pandas` методов:

```
hist = anime['rating'].hist()
figure = hist.get_figure()
```

Результат выполнения:



Полученное распределение похоже на нормальное. Явных выбросов мы не наблюдаем. Поэтому DataFrame `anime` по качеству данных нас устраивает.

Проверим DataFrame `rating` на наличие пропусков:

```
rating.isnull().mean().sort_values(ascending=False)
```

Результат выполнения:

```
user_id     0.0
anime_id    0.0
rating      0.0
dtype: float64
```

Посмотрим на числовые показатели. Минимальное значение:

```
rating.min()
```

Результат выполнения:

```
user_id     1
anime_id    1
rating     -1
dtype: int64
```

Максимальное значение:

```
rating.max()
```

Результат выполнения:

```
user_id     73516
anime_id    34519
rating         10
dtype: int64
```

Среднее значение:

```
rating.mean()
```

Результат выполнения:

```
user_id     36727.956745
anime_id     8909.072104
rating          6.144030
dtype: float64
```

Медианное значение:

```
rating.median()
```

Результат выполнения:

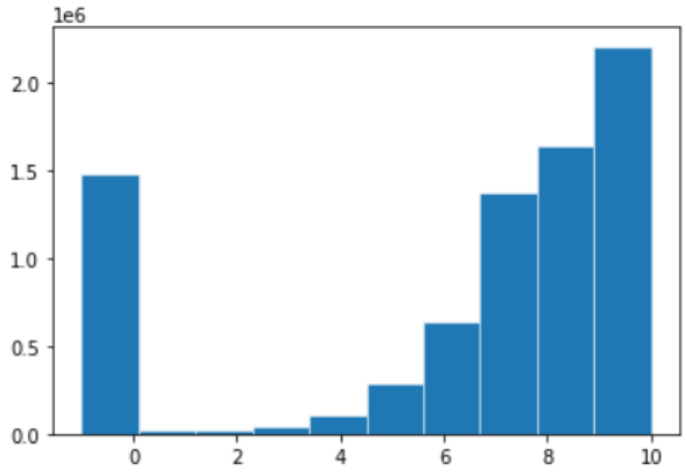
```
user_id     36791.0
anime_id     6213.0
rating          7.0
dtype: float64
```

Посмотрев на полученные результаты, мы видим, что явных выбросов нет.

Построим гистограмму частот с помощью библиотеки `matplotlib`:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.hist(rating['rating'], linewidth=0.5, edgecolor="white")
plt.show()
```

Результат выполнения:



Распределение рейтинга не выглядит нормальным: имеется столбец соответствующий пользователям, которые не поставили оценку. Также виден перекос в сторону больших оценок. То есть если пользователи оценивали аниме, то они оценивали его хорошо.

Поскольку распределения рейтинга не имеет нормального вида, это повод задуматься, почему так происходит. Мы можем предположить, что у пользователей есть свойство ставить оценку только тогда, когда им что-то понравилось. Если пользователю не понравилось аниме, он не станет досматривать, и скорее всего, оценку не поставит. Это объясняет большую долю аниме с оценкой «–1».

Вернемся к работе с DataFrame `anime`. Поскольку изначально в поставленной задаче нас интересуют только хорошие аниме, нам нужно понимать, какой у них рейтинг. Поэтому записи без значения рейтинга мы можем отбросить для нашей задачи. В другом случае мы могли бы заполнить эти значения каким-либо образом — например, средними или медианными значениями.

Отбросим пропуски:

```
anime = anime.dropna(subset=['rating'])
```

Проверим, получилось ли у нас:

```
anime.isnull().mean().sort_values(ascending=False)
```

Результат выполнения:

```
genre        0.003896
anime_id     0.000000
name         0.000000
type         0.000000
episodes     0.000000
rating       0.000000
members      0.000000
dtype: float64
```

Видно, что пропуски остались только в колонке «жанр». Вычистив пропуски в колонке «рейтинг», мы избавились от пропусков в колонке «тип».

Посмотрим сводную статистику:

```
anime.describe()
```

Результат выполнения:

	anime_id	rating	members
count	12064.000000	12064.000000	1.206400e+04
mean	13704.476044	6.473902	1.827952e+04
std	11260.369521	1.026746	5.527578e+04
min	1.000000	1.670000	1.200000e+01
25%	3409.250000	5.880000	2.210000e+02
50%	10004.000000	6.570000	1.539000e+03
75%	23863.500000	7.180000	9.485500e+03
max	34519.000000	10.000000	1.013917e+06

Каких-либо перекосов в данных мы не наблюдаем.

На всякий случай вычистим дубликаты в обоих DataFrame. Это необходимо сделать, чтобы не было перекосов в данных. Такое может наблюдаться, если замножились записи по какому-то конкретному аниме. Подобное засорит описательную статистику набора данных.

```
anime.drop_duplicates(inplace=True)
rating.drop_duplicates(inplace=True)
```

Перейдем к реализации поставленной перед нами задачи. Пусть первое, что нам необходимо определить — уникальное количество аниме по типам (TB, фильм и т. п.):

```
anime['type'].value_counts()
```

Результат выполнения: самый популярный тип — ТВ-шоу:

```
TV         3671
OVA        3285
Movie      2297
Special    1671
ONA         652
Music       488
Name: type, dtype: int64
```

Посмотрим по долям. Чтобы получить именно доли, мы нормализуем наши значения:

```
anime['type'].value_counts(normalize=True)
```

Результат выполнения:

```
TV         0.304294
OVA        0.272298
Movie      0.190401
Special    0.138511
ONA        0.054045
Music      0.040451
Name: type, dtype: float64
```

Следующим шагом мы сгруппируем аниме по типу и жанру и посмотрим, какое общее количество записей мы имеем:

```
anime.groupby('type')['genre'].value_counts()
```

Результат выполнения:

```
type   genre
Movie  Dementia                                 124
       Comedy                                    85
       Kids                                      72
       Drama                                     54
       Fantasy                                   53
                                               ...
TV     Sci-Fi, Thriller                           1
       Seinen, Slice of Life, Supernatural        1
       Seinen, Sports                             1
       Shoujo                                     1
       Shounen, Supernatural                      1
Name: genre, Length: 4885, dtype: int64
```

Теперь, когда мы получили количество записей по конкретным жанрам и типам, можем перейти к получению их средних оценок:

```
anime.groupby(['type','genre'])['rating'].mean()
```

Результат выполнения:

```
type   genre
Movie  Action
5.389444
       Action, Adventure
5.537500
       Action, Adventure, Cars, Comedy, Sci-Fi, Shounen
6.850000
       Action, Adventure, Cars, Sci-Fi
6.860000
       Action, Adventure, Comedy
5.937500
                                                                    ...
TV     Shounen, Sports, Super Power
7.400000
       Shounen, Supernatural
7.690000
       Slice of Life
5.956000
       Sports
6.601875
       Supernatural
5.995000
Name: rating, Length: 4885, dtype: float64
```

Следующим этапом мы можем выгрузить оба полученных набора данных в csv, объединить их и передать заказчику. То есть дать понять, какие типы жанров и какие типы аниме лучше всего привлекать.

Для передачи заказчику очень важно вывести среднюю оценку и дополнить ее информацией о количестве этих оценок. Если мы будем понимать, что оценок определенного жанра и типа достаточно много, и оценка является высокой, то такой жанр и тип аниме стоит подписывать в первую очередь.

2. Пример анализа данных из нескольких источников

Теперь рассмотрим пример, когда у нас есть несколько источников данных. В предыдущем примере мы подгружали два DataFrame. В нашем примере они имеют один источник, но это два разных набора данных.

Давайте представим, что у нас имеется два набора данных из разных источников.

Поскольку данные мы подгрузили ранее, сейчас дополнительно подгружать csv-файлы нам не нужно.

Объединим два DataFrame с помощью `merge()`:

```
data = rating.merge(anime, left_on='anime_id',
right_on='anime_id', suffixes=('_rating', '_anime'))
data
```

Объединение выполняем по полю `anime_id`, которое присутствует в обоих DataFrame. Чтобы понимать, какое из полей `rating` относится к DataFrame `rating`, а какое к `anime`, мы добавляем суффиксы.

Результат выполнения:

	user_id	anime_id	rating_rating	name	genre	type	episodes	rating_anime	members
0	1	20	-1	Naruto	Action, Comedy, Martial Arts, Shounen, Super P...	TV	220	7.81	683297
1	3	20	8	Naruto	Action, Comedy, Martial Arts, Shounen, Super P...	TV	220	7.81	683297
2	5	20	6	Naruto	Action, Comedy, Martial Arts, Shounen, Super P...	TV	220	7.81	683297
3	6	20	-1	Naruto	Action, Comedy, Martial Arts, Shounen, Super P...	TV	220	7.81	683297
4	10	20	-1	Naruto	Action, Comedy, Martial Arts, Shounen, Super P...	TV	220	7.81	683297
...	...	...	...	...	...	...	...	...	...
7813715	65682	30450	8	Dr. Slump: Hoyoyo! Arale no Himitsu Dai Koukai...	Comedy, Sci-Fi, Shounen	Special	1	6.17	248
7813716	69497	33484	10	Shiroi Zou	Action, Historical, Kids	Movie	1	4.71	45
7813717	70463	29481	-1	Kakinoki Mokkii	Fantasy, Kids	Special	1	4.33	61
7813718	72404	34412	-1	Hashiri Hajimeta bakari no Kimi ni	Music	Music	1	6.76	239
7813719	72800	30738	4	Gamba: Gamba to Nakama-tachi	Adventure, Kids	Movie	1	5.55	185

7813720 rows × 9 columns

Как видно, у нас есть три колонки из DataFrame `rating`, все остальные колонки — из `anime`. При этом видим, что наши суффиксы сработали, мы можем явным образом отличить рейтинги из разных наборов данных.

Посмотрим некоторые статистические выкладки по полученной таблице. Посчитаем количество записей по конкретным названиям аниме, отсортируем их по убыванию:

```
data['name'].value_counts(ascending=False)
```

Результат выполнения:

```
Death Note                                39340
Sword Art Online                          30583
Shingeki no Kyojin                        29583
```

```
Code Geass: Hangyaku no Lelouch                 27718
Elfen Lied                                      27506
                                                  ...
Paboo &amp;amp; Mojies                              1
Ponta to Ensoku                                     1
Kono Shihai kara no Sotsugyou: Ozaki Yutaka         1
Tsuki ga Noboru made ni                             1
Gamba: Gamba to Nakama-tachi                        1
Name: name, Length: 11193, dtype: int64
```

В данном случае больше всего записей, то есть оценок пользователей по аниме `Death Note`.

Произведем похожую операцию — сгруппируем значения среднего рейтинга из таблицы по рейтингу (в которой встречаются конкретные оценки каждого пользователя по конкретному аниме, у каждого пользователя может быть множество оценок) по названию конкретного аниме:

```
data.groupby(['name'])['rating_rating'].mean()
```

Результат выполнения:

```
name
&quot;0&quot;                                                 2.769231
&quot;Aesop&quot; no Ohanashi yori: Ushi to Kaeru, Yokubatta Inu    0.000000
&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hatsukoi                 5.774936
&quot;Bungaku Shoujo&quot; Memoire                                  6.155748
&quot;Bungaku Shoujo&quot; Movie                                    6.457980
                                                                     ...
xxxHOLiC Kei                                                    6.720774
xxxHOLiC Movie: Manatsu no Yoru no Yume                         6.313742
xxxHOLiC Rou                                                    6.403173
xxxHOLiC Shunmuki                                               6.238602
◯                                                                3.000000
Name: rating_rating, Length: 11193, dtype: float64
```

Мы получили набор названий аниме и некоторый средний рейтинг.

Для примера давайте сравним среднюю оценку по аниме Naruto:

```
data[data.name == 'Naruto'].groupby(['name'])['rating_rating'].mean()
```

Результат выполнения:

```
name
Naruto    6.571726
Name: rating_rating, dtype: float64
```

Мы получили, что средняя оценка, если брать оценки пользователя из DataFrame `rating`, будет 6,57. Если мы посмотрим на первоначальные данные, то увидим, что оценка отличается.

То есть если это два разных источника информации, то тем самым мы получили две разные оценки по одному и тому же аниме. Мы можем выводить обе оценки, а можем придумать какой-либо коэффициент (например, если одному из источников мы доверяем больше). Дальнейшие действия зависят от конкретной задачи.

Мы видим, что оценки не сходятся. На это может влиять оценка «–1» в рейтинге, если вычистить такие значения, возможно, данные из разных источников будут одинаковые.

Перейдем к финальному шагу.

Сделаем агрегацию по названию аниме с количеством поставивших оценку пользователей и медианным рейтингом. Группируем по названию аниме, прописываем, что нас интересует агрегация:

```
df = data.groupby(['name']).agg({'rating_rating': ['median'], 'user_id':
['count']}).reset_index()
df
```

Результат работы:

	name	rating_rating	user_id
		median	count
0	"0"	4.5	26
1	"Aesop" no Ohanashi yori: Ushi to Ka...	0.0	2
2	"Bungaku Shoujo" Kyou no Oyatsu: Hat...	7.0	782
3	"Bungaku Shoujo" Memoire	7.0	809
4	"Bungaku Shoujo" Movie	8.0	1535
...	...	...	...
11188	xxxHOLiC Kei	8.0	3413
11189	xxxHOLiC Movie: Manatsu no Yoru no Yume	8.0	2365
11190	xxxHOLiC Rou	8.0	1513
11191	xxxHOLiC Shunmuki	8.0	1974
11192	○	2.0	6

11193 rows × 3 columns

Мы получили некую сводную таблицу, где содержится медианная оценка определенного аниме, а также количество оценок.

Далее полученный набор данных можно выгрузить в csv-файл и передать заказчику.

В целом по полученному набору данных мы можем выделить аниме, которые имеют высокий рейтинг и много оценок. Соответственно, такое аниме нужно подписывать в первую очередь.

3. Работа с матрицей корреляции

Будем использовать DataFrame `anime` из предыдущего задания. Подгрузим его и еще раз посмотрим на структуру:

```
anime = pd.read_csv('anime.csv', sep=',')
anime.head()
```

Результат выполнения:

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

Введем базовую теорию, которую будем использовать в дальнейшем.

Коэффициент корреляции — это статистическая мера, которая вычисляет силу связи между относительными движениями двух переменных. Значения коэффициента корреляции находятся в диапазоне от –1.0 до 1.0.

Основные выводы в зависимости от коэффициента корреляции:

1. Значения всегда находятся в диапазоне от -1 (сильная отрицательная связь) до +1 (сильная положительная связь).
2. Значения при нулевом значении или близкие к нулю подразумевают слабую или отсутствующую связь.
3. Значения коэффициентов корреляции менее +0,8 или более -0,8 не считаются значимыми.

Построим матрицу корреляции:

```
anime.corr()
```

Результат выполнения:

	anime_id	rating	members
anime_id	1.000000	-0.284625	-0.080071
rating	-0.284625	1.000000	0.387979
members	-0.080071	0.387979	1.000000

Видно, что диагональ матрицы равна «1», поскольку одинаковые значения имеют сильную положительную связь. Кроме этого, сильной значимой корреляции между разными параметрами не наблюдается.

Для удобства коэффициенты корреляции можно округлить:

```
anime.corr().round(2)
```

Результат выполнения — полученную матрицу корреляции удобнее воспринимать и читать:

	anime_id	rating	members
anime_id	1.00	-0.28	-0.08
rating	-0.28	1.00	0.39
members	-0.08	0.39	1.00

Следующим шагом визуализируем матрицу корреляции:

```
corr = anime.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Визуализация позволяет наглядно представить высокие и низкие значения коэффициента корреляции.

Результат:

	anime_id	rating	members
anime_id	1.000000	-0.284625	-0.080071
rating	-0.284625	1.000000	0.387979
members	-0.080071	0.387979	1.000000

Существует еще один вариант визуализации, который отличается паттерном представления цвета:

```
corr = anime.corr()
corr.style.background_gradient(cmap='RdYlGn')
```

Результат выполнения:

	anime_id	rating	members
anime_id	1.000000	-0.284625	-0.080071
rating	-0.284625	1.000000	0.387979
members	-0.080071	0.387979	1.000000

В рассматриваемом случае мы делаем вывод, что какой-либо сильной корреляции в данных не наблюдается. Если бы она была, мы могли упростить наши выводы.

Например, если между рейтингом и количеством подписчиков есть сильная корреляция, то мы могли бы потенциально сделать вывод о том, какое аниме нам стоит подписывать по количеству подписчиков, и при этом не принимать во внимание значение рейтинга.

Но по данным примерам мы такой вывод сделать не можем. А значит, упростить нашу задачу мы тоже не можем.

4. Создание интерактивных графиков

В команде не всегда бывает готовый сервис для визуализации, поэтому Python в данном случае может пригодиться. В Python есть достаточно сильные сервисы, которые помогают анимировать график, добавить всплывающие фильтры и т. д.

Интерактивный график представляет собой график, в который можно добавить слайдер, фильтры и можно динамически изменять.

Для начала добавим возможность отображать интерактивные графики:

```
%matplotlib notebook
```

Данный пример мы демонстрируем в Google Collab, в нем есть свои особенности отображения интерактивных графиков. Чтобы отобразить интерактивный график, дополнительно требуется импортировать блок `rc` из библиотеки `matplotlib` и `HTML` из `IPython.display`.

В конце кода мы должны прописать стоки:

```
rc('animation', html='jshtml')
animation
```

Если бы мы работали в JupyterHub, было бы достаточно прописать `plt.show`.

Создадим простую анимацию обычного линейного графика:

```
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib import rc
import numpy as np
from IPython.display import HTML

x = []
y = []

figure, ax = plt.subplots()

# Задаем границы для осей x и y
ax.set_xlim(0, 100)
ax.set_ylim(0, 10)

line,  = ax.plot(0, 0)
```

```
def animation_function(i):
    x.append(i * 10)
    y.append(i)

    line.set_xdata(x)
    line.set_ydata(y)
    return line,

animation = FuncAnimation(figure,
                          func = animation_function,
                          frames = np.arange(0, 10, 0.1),
                          interval = 10)
rc('animation', html='jshtml')
animation
```

В построенном линейном графике `x` будет принимать значения от 0 до 100, `y` — от 0 до 10. Начальное состояние графика задается строкой — точка (0, 0):

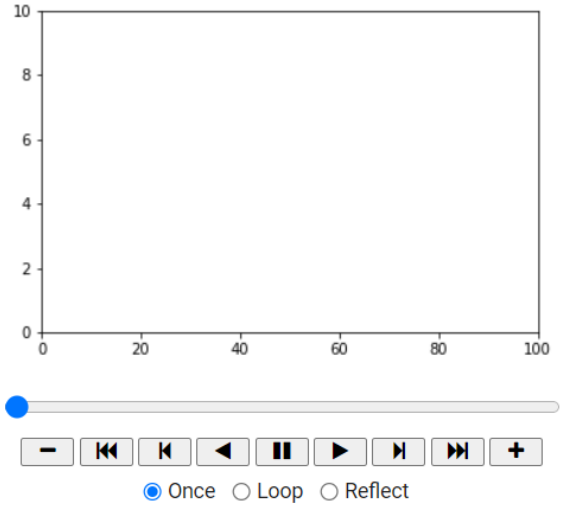
```
line,  = ax.plot(0, 0)
```

Функция `animation_function()` — функция анимации, где с каждым шагом значение `x` умножается на 10 (0, 10, 20, 30 и т. д.), а значение `y` линейно возрастает (0, 1, 2, 3 и т. д.).

В следующей строке для нашей анимации мы прописываем некоторые параметры. Например, интервал:

```
animation = FuncAnimation(figure,
                          func = animation_function,
                          frames = np.arange(0, 10, 0.1),
                          interval = 10)
```

В Google Collab окно для графика будет выглядеть следующим образом:



Чтобы график был построен, необходимо нажать Play. Анимацией можно управлять (нажать паузу, отмотать назад) с помощью соответствующих кнопок.

Рассмотрим анимацию для bar chart:

```
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation, writers
import numpy as np

fig = plt.figure(figsize = (7,5))
axes = fig.add_subplot(1,1,1)
axes.set_ylim(0, 300)
palette = ['red', 'orange', 'yellow',
           'green', 'blue', 'purple']

y1, y2, y3, y4, y5, y6 = [], [], [], [], [], []

def animation_function(i):
    y1 = i
    y2 = 3 * i
    y3 = 2 * i
    y4 = 6 * i
    y5 = 5 * i
    y6 = 4 * i

    plt.xlabel("User ID")
    plt.ylabel("Money spent, rub")

    plt.bar(["user_1", "user_2", "user_3",
             "user_4", "user_5", "user_6"],
            [y1, y2, y3, y4, y5, y6],
```

```
            color = palette)

plt.title("Bar Chart Animation")

animation = FuncAnimation(fig, animation_function,
                          interval = 50)
rc('animation', html='jshtml')
animation
```

В коде мы подключаем те же самые библиотеки, что и в примере с линейным графиком. Дополнительно устанавливаем цветовую палитру:

```
palette = ['red', 'orange', 'yellow',
           'green', 'blue', 'purple']
```

В функции `animation_function()` мы прописываем, с какой скоростью столбцы будут расти.

Задаем в коде названия осей:

```
plt.xlabel("User ID")
plt.ylabel("Money spent, rub")
```

В примере сумма платежей (ось `y`) растет по определенному закону, но если бы мы работали с реальными данными, мы могли бы выгружать значения и анимировать их.

Название графика задается с помощью строки:

```
plt.title("Bar Chart Animation")
```

Рассмотрим еще одну библиотеку для визуализации данных — `plotly`. Начнем работу с того, что подгрузим все необходимые библиотеки:

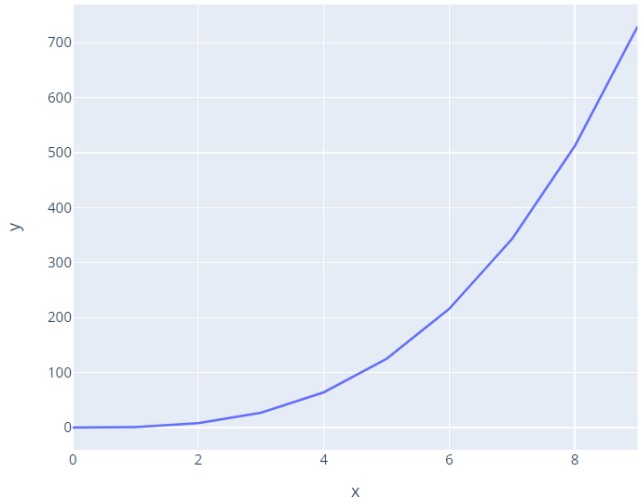
```
import plotly
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots

import numpy as np
import pandas as pd
```

Построим обычный график по точкам для функции $y = x^3$:

```
x = np.arange(0, 10, 0.5)
def f(x):
    return x**3
```

```
px.scatter(x=x, y=f(x)).show()
```

Результат построения графика:



Если навести мышью на график, мы увидим координаты точек. Также мы можем увеличить («зумировать») интересующий нас участок графика,

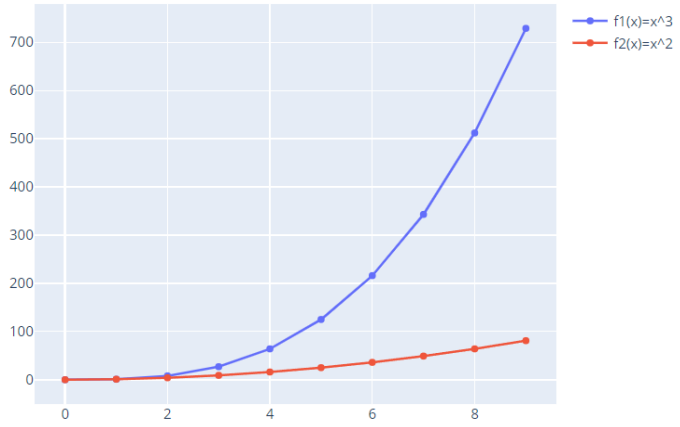
Построим на одном графике две функции. Для удобства мы подпишем обе функции:

```
fig = go.Figure()
fig.add_trace(go.Line(x=x, y=f(x), name='f1(x)=x^3'))
fig.add_trace(go.Line(x=x, y=x*x, name='f2(x)=x^2'))
fig.show()
```

Теперь мы можем сравнивать два графика. Это бывает полезно, например, мы таким образом можем сравнить показатели по выручке за текущий год и за аналогичный период предыдущего года.

Важно отметить, что все графики, построенные с помощью библиотеки `plotly`, являются интерактивными. На графиках подсвечиваются значения точек при наведении, можно увеличивать и уменьшать масштаб, выделять определенную область построения.

Результат построения:

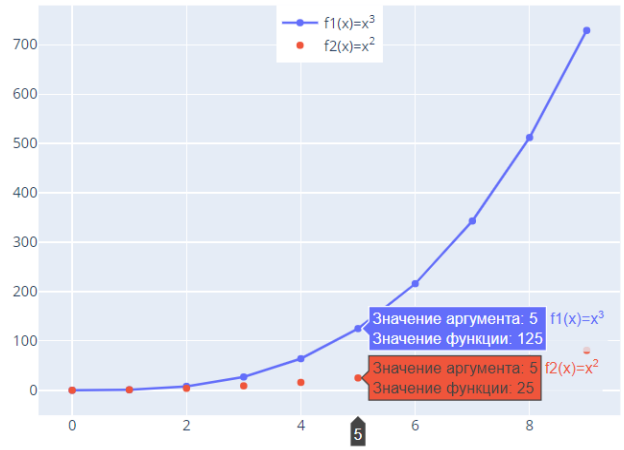


Теперь давайте усложним нашу конструкцию.

Добавим всплывающие названия, изменим режим отображения графика, изменим ориентацию легенды:

```
fig = go.Figure()
fig.add_trace(go.Line(x=x, y=f(x), mode='lines+markers',
name='f1(x)=x<sup>3</sup>'))
fig.add_trace(go.Line(x=x, y=x*x, mode='markers',
name='f2(x)=x<sup>2</sup>'))
fig.update_layout(legend_orientation="v",
                  legend=dict(x=0.5, xanchor="auto"),
                  hovermode = 'x')
fig.update_traces(hoverinfo="all", hovertemplate="Значение
аргумента: %{x}<br>Значение функции: %{y}")
fig.show()
```

Результат построения:



Добавим для графика слайдер:

```
ticks_number = len(x)
trace_list = [go.Line(visible=True, x=[x[0]], y=[f(x)[0]],
mode='lines+markers', name='f(x)=x<sup>3</sup>')]

for i in range(1, len(x)):
    trace_list.append(go.Line(visible=False, x=x[:i+1],
y=f(x[:i+1]), mode='lines+markers', name='f(x)=x<sup>3</sup>'))

fig = go.Figure(data=trace_list)

steps = []
for i in range(ticks_number):
    step = dict(
        method = 'restyle',
        args = ['visible', [False] * len(fig.data)],
    )
    step['args'][1][i] = True

    steps.append(step)

sliders = [dict(
    steps = steps,
)]

fig.layout.sliders = sliders

fig.show()
```

Слайдер представляет собой бегунок, перемещая который мы можем двигать значения на графике. В примере значения названы как step-0, step-1 и т. д. Но мы

можем заменить их, например, на конкретные дни и использовать в графиках для выделения определенного промежутка.

В коде выше мы задали график, выбрали режим (линии и точки). Далее прописали, что нас интересуют слайдеры:

```
sliders = [dict(
    steps = steps,
)]
```

Слайдеры будут учитываться по шагам, это также отражено в цикле `for` кода:

```
step['args'][1][i] = True
steps.append(step)
```

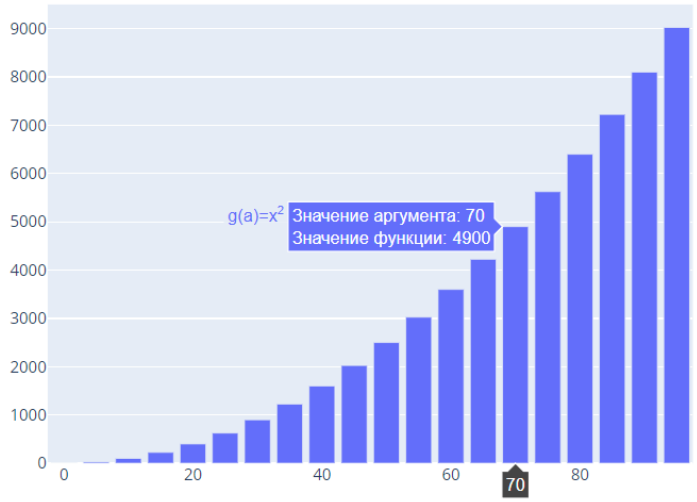
Пример построения интерактивных гистограмм. С такой гистограммой мы также можем интерактивно работать. Мы задаем новую функцию, в примере это будет квадратичная функция:

```
a = np.arange(0, 100, 5.0)
def g(a):
    return a**2

px.bar(x=a, y=g(a)).show()
```

Усложним конструкцию, добавив легенду и всплывающие подсказки для каждого значения функции:

```
fig = go.Figure()
fig.add_trace(go.Bar(x=a, y=g(a),  name='g(a)=x<sup>2</sup>'))
fig.update_layout(legend_orientation="h",
                  legend=dict(x=0.5, xanchor="center"),
                  hovermode = 'x')
fig.update_traces(hoverinfo="all", hovertemplate="Значение
аргумента: %{x}<br>Значение функции: %{y}")
fig.show()
```

Результат построения:



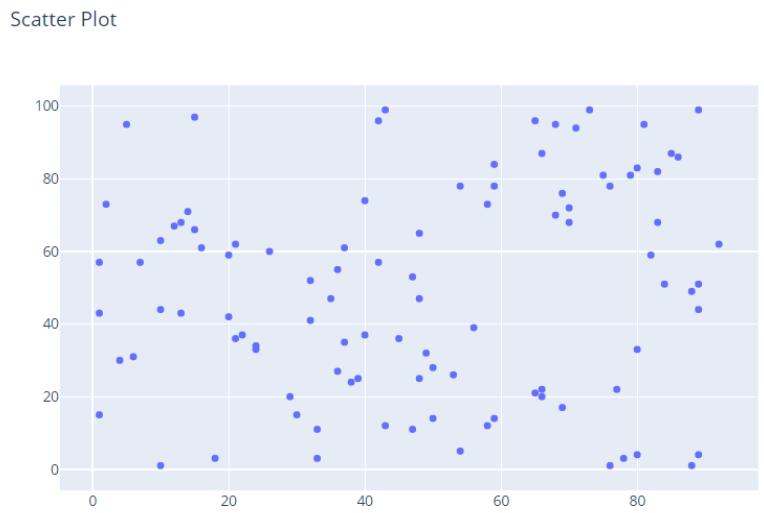
Добавим на график выпадающий список. Для начала подключим необходимые библиотеки:

```
import plotly.graph_objects as px
import numpy as np
```

Построим обычный график по точкам. Мы генерируем случайные значения и визуализируем их на графике:

```
np.random.seed(0)

random_x = np.random.randint(1, 100, 100)
random_y = np.random.randint(1, 100, 100)

fig = go.Figure(data=[px.Scatter(
        x=random_x,
        y=random_y,
        mode='markers',)
])

fig.update_layout(title = "Scatter Plot")

fig.show()
```

Результат построения:

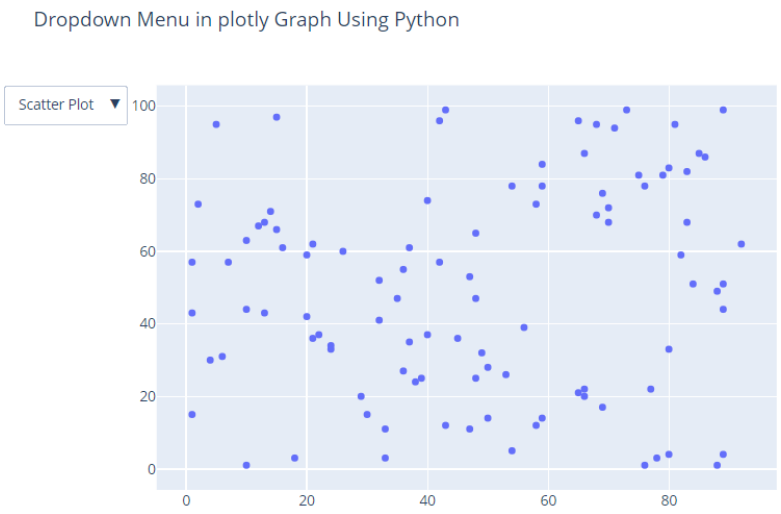


Усложним написанную программу и добавим выпадающий список. В списке мы можем выбирать желаемый тип графика:

```
np.random.seed(0)

random_x = np.random.randint(1, 100, 100)
random_y = np.random.randint(1, 100, 100)

fig = px.Figure(data=[px.Scatter(
    x=random_x,
    y=random_y,
    mode='markers',)
])

fig.update_layout(
    updatemenus=[
        dict(
        buttons=list([
            dict(
            args=["type", "scatter"],
            label="Scatter Plot",
            method="restyle"
            ),
                        dict(
                        args=["type", "box"],
                        label="Box Plot",
                        method="restyle"
                        ),
```

```
                dict(
                args=["type", "bar"],
                label="Bar Plot",
                method="restyle"
                )
                ]),
                direction="down",
                ),
            ],
        title = "Dropdown Menu in plotly Graph Using Python"
)

fig.show()
```

Результат построения:



В выпадающем списке мы можем выбирать вид графика.

Аналогичным способом мы можем добавлять фильтры на график для удобства представления информации и выделения конкретных данных. Подобный функционал часто встречается в уже готовых средствах визуализации, но мы можем реализовать его и с помощью Python.

Таким образом, мы разобрали несколько приемов, которые могут быть полезны. Аналогичных приемов достаточно много, и если Python станет вашим основным средством визуализации, вам придется дополнительно осваивать подобные инструменты.

5. Категоризация на примере анализа данных электроавтомобилей

Задача заключается в подготовке данных по электроавтомобилям для дальнейшего машинного обучения.

Поскольку мы работаем в Google Colab, присоединимся к диску и выберем каталог для работы:

```
from google.colab import drive
drive.mount('/content/drive')

import os
print("current path", os.getcwd())
os.chdir('/content/drive/MyDrive/Advanced ML course/Module
2/video/input')
print("current path", os.getcwd())
```

Загрузим данные электрокаров:

```
import pandas as pd
import numpy as np
EC = pd.read_csv('Electric Car.csv', sep=',')
EVP = pd.read_csv('Electric Vehicle Population Data.csv', sep=',')
```

Мы загрузили два файла. Первый файл — информация о типах автомобилей, второй — данные конкретных автомобилей с идентификационным номером.

Посмотрим на структуру данных:

```
EC.head()
EVP.head()
```

Рассмотрим датасет, содержащий информацию о типах автомобилей. В датасете присутствуют следующие колонки: бренд, модель, скорость и другие.

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	RapidCharge	PowerTrain	PlugType	BodyStyle	Segment	Seats	PriceEuro
0	Tesla	Model 3 Long Range Dual Motor	4.6	233	450	161	940	Yes	AWD	Type 2 CCS	Sedan	D	5	55480
1	Volkswagen	ID.3 Pure	10.0	160	270	167	250	Yes	RWD	Type 2 CCS	Hatchback	C	5	30000
2	Polestar	2	4.7	210	400	181	620	Yes	AWD	Type 2 CCS	Liftback	D	5	56440
3	BMW	iX3	6.8	180	360	206	560	Yes	RWD	Type 2 CCS	SUV	D	5	68040
4	Honda	e	9.5	145	170	168	190	Yes	RWD	Type 2 CCS	Hatchback	B	4	32997

Второй загруженный набор данных:

	VIN (1-10)	County	City	State	ZIP Code	Model Year	Make	Model	Electric Vehicle Type	Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID	Vehicle Location	Electric Utility
0	1N4AZ0CP3E	King	RENTON	WA	98059	2014	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	84	0	5.0	250845815	POINT (-122.132064 47.494834)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
1	1N4AZ1CP2J	King	REDMOND	WA	98053	2018	NISSAN	LEAF	Battery Electric Vehicle (BEV)	Clean Alternative Fuel Vehicle Eligible	151	0	45.0	309178936	POINT (-122.024951 47.670286)	PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
2	WBY1Z8C50H	King	SEATTLE	WA	98125	2017	BMW	I3	Plug-in Hybrid Electric Vehicle (PHEV)	Clean Alternative Fuel Vehicle Eligible	97	0	46.0	8751711	POINT (-122.303604 47.716244)	CITY OF SEATTLE - (WA)||CITY OF TACOMA - (WA)

Посмотрим, сколько данных содержат датасеты. Для этого используем команду `shape`:

```
EC.shape
EVP.shape
```

Для первого набора данных получаем 14 признаков и 103 строки. Второй набор содержит 16 признаков и 96961 строку.

Посмотрим основную информацию с помощью команды `info()`:

```
EC.info()
EVP.info()
```

Видим количество данных и их тип для первого датафрейма:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Brand            103 non-null    object
 1   Model            103 non-null    object
 2   AccelSec         103 non-null    float64
 3   TopSpeed_KmH     103 non-null    int64
 4   Range_Km         103 non-null    int64
 5   Efficiency_WhKm  103 non-null    int64
 6   FastCharge_KmH   103 non-null    object
 7   RapidCharge      103 non-null    object
 8   PowerTrain       103 non-null    object
 9   PlugType         103 non-null    object
 10  BodyStyle        103 non-null    object
 11  Segment          103 non-null    object
 12  Seats            103 non-null    int64
 13  PriceEuro        103 non-null    int64
dtypes: float64(1), int64(5), object(8)
memory usage: 11.4+ KB
```

Для второго датафрейма:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96961 entries, 0 to 96960
Data columns (total 16 columns):
 #   Column                                             Non-Null Count  Dtype
---  ------                                             --------------  -----
 0   VIN (1-10)                                         96961 non-null  object
 1   County                                             96954 non-null  object
 2   City                                               96961 non-null  object
 3   State                                              96961 non-null  object
 4   ZIP Code                                           96961 non-null  int64
 5   Model Year                                         96961 non-null  int64
 6   Make                                               96961 non-null  object
 7   Model                                              96961 non-null  object
 8   Electric Vehicle Type                              96961 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  96961 non-null  object
 10  Electric Range                                     96961 non-null  int64
 11  Base MSRP                                          96961 non-null  int64
 12  Legislative District                               96710 non-null  float64
 13  DOL Vehicle ID                                     96961 non-null  int64
 14  Vehicle Location                                   96954 non-null  object
 15  Electric Utility                                   95869 non-null  object
dtypes: float64(1), int64(5), object(10)
memory usage: 11.8+ MB
```

Посчитаем, сколько содержится уникальных значений идентификатора `DOL Vehicle ID`:

```
EVP['DOL Vehicle ID'].nunique()
```

Получаем 96961.

Проверим, сколько уникальных городов:

```
EVP['City'].nunique()
```

Получаем 624.

Проанализируем более подробно числовые данные датафрейма `EC`:

```
EC.describe()
```

Таким образом, можем проанализировать показатели по цене, скорости и другие:

	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro
count	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000
mean	7.396117	179.194175	338.786408	189.165049	4.883495	55811.563107
std	3.017430	43.573030	126.014444	29.566839	0.795834	34134.665280
min	2.100000	123.000000	95.000000	104.000000	2.000000	20129.000000
25%	5.100000	150.000000	250.000000	168.000000	5.000000	34429.500000
50%	7.300000	160.000000	340.000000	180.000000	5.000000	45000.000000
75%	9.000000	200.000000	400.000000	203.000000	5.000000	65000.000000
max	22.400000	410.000000	970.000000	273.000000	7.000000	215000.000000

Аналогично посмотрим второй датафрейм:

```
EC.describe()
```

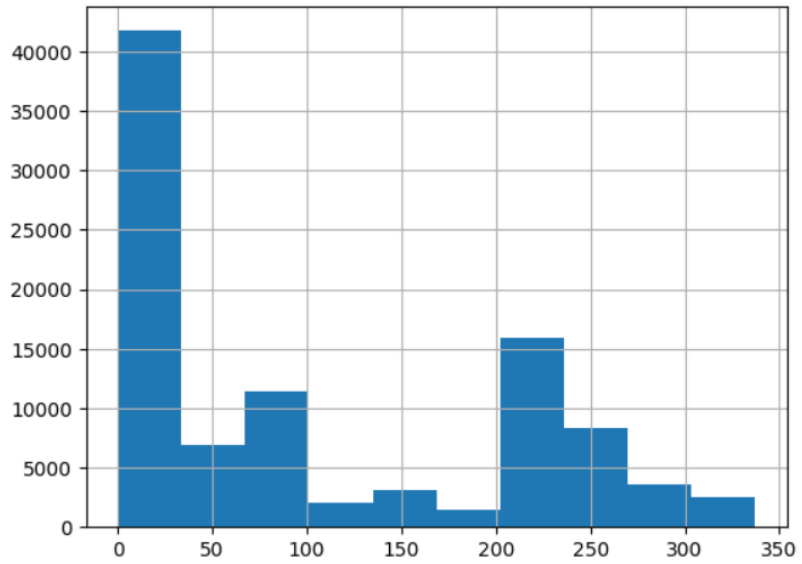
В результате можем оценить год выпуска автомобиля, электрический запас хода:

	ZIP Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID
count	96961.000000	96961.000000	96961.000000	96961.00000	96710.000000	9.696100e+04
mean	98145.294768	2018.367622	102.221996	2400.14676	29.917030	1.970336e+08
std	2820.876011	2.767345	103.862590	11980.87321	14.654592	1.044936e+08
min	745.000000	1993.000000	0.000000	0.00000	0.000000	4.385000e+03
25%	98052.000000	2017.000000	0.000000	0.00000	19.000000	1.388449e+08
50%	98121.000000	2019.000000	58.000000	0.00000	34.000000	1.790813e+08
75%	98370.000000	2021.000000	215.000000	0.00000	43.000000	2.211784e+08
max	99801.000000	2022.000000	337.000000	845000.00000	49.000000	4.792548e+08

Построим гистограмму по электрическому диапазону:

```
hist = EVP['Electric Range'].hist()
figure = hist.get_figure()
```

Видно, что большее количество автомобилей попадает в диапазон до 50 км:



Так как в обеих таблицах присутствуют одинаковые столбцы `Brand` и `Make`, переведем все значения этих столбцов в верхний регистр. Таким образом, написания значений будут совпадать:

```
EC.Brand = EC.Brand.str.upper()
EVP.Make = EVP.Make.str.upper()
```

Затем заменим название столбца `Make` на `Brand`. В результате столбцы в двух таблицах будут называться одинаково:

```
EVP.rename(columns = {'Make':'Brand'}, inplace = True )
```

Объединим две таблицы с помощью `merge()`:

```
data = EC.merge(EVP, how = 'outer')
```

Построим корреляционную матрицу:

```
data.corr().round(2)
```

Нужно учесть, что корреляция строится только для числовых данных. В полученной матрице корреляции присутствуют значения `NaN`, но тем не менее мы можем делать некоторые выводы:

	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro	ZIP Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID
AccelSec	1.00	-0.79	-0.68	-0.38	-0.18	-0.63	NaN	NaN	NaN	NaN	NaN	NaN
TopSpeed_KmH	-0.79	1.00	0.75	0.36	0.13	0.83	NaN	NaN	NaN	NaN	NaN	NaN
Range_Km	-0.68	0.75	1.00	0.31	0.30	0.67	NaN	NaN	NaN	NaN	NaN	NaN
Efficiency_WhKm	-0.38	0.36	0.31	1.00	0.30	0.40	NaN	NaN	NaN	NaN	NaN	NaN
Seats	-0.18	0.13	0.30	0.30	1.00	0.02	NaN	NaN	NaN	NaN	NaN	NaN
PriceEuro	-0.63	0.83	0.67	0.40	0.02	1.00	NaN	NaN	NaN	NaN	NaN	NaN
ZIP Code	NaN	NaN	NaN	NaN	NaN	NaN	1.00	-0.00	-0.01	-0.00	-0.44	-0.00
Model Year	NaN	NaN	NaN	NaN	NaN	NaN	-0.00	1.00	-0.13	-0.22	0.02	-0.11
Electric Range	NaN	NaN	NaN	NaN	NaN	NaN	-0.01	-0.13	1.00	0.06	0.04	0.04
Base MSRP	NaN	NaN	NaN	NaN	NaN	NaN	-0.00	-0.22	0.06	1.00	0.01	0.01
Legislative District	NaN	NaN	NaN	NaN	NaN	NaN	-0.44	0.02	0.04	0.01	1.00	-0.00
DOL Vehicle ID	NaN	NaN	NaN	NaN	NaN	NaN	-0.00	-0.11	0.04	0.01	-0.00	1.00

Для удобного анализа матрицы корреляции добавим цвет:

```
corr = data.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Видно, что самая большая корреляция между ценой, скоростью и диапазоном:

	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro	ZIP Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID
AccelSec	1.000000	-0.786195	-0.677062	-0.382904	-0.175335	-0.627174	nan	nan	nan	nan	nan	nan
TopSpeed_KmH	-0.786195	1.000000	0.746662	0.355675	0.126470	0.829057	nan	nan	nan	nan	nan	nan
Range_Km	-0.677062	0.746662	1.000000	0.313077	0.300163	0.674844	nan	nan	nan	nan	nan	nan
Efficiency_WhKm	-0.382904	0.355675	0.313077	1.000000	0.301230	0.396705	nan	nan	nan	nan	nan	nan
Seats	-0.175335	0.126470	0.300163	0.301230	1.000000	0.020920	nan	nan	nan	nan	nan	nan
PriceEuro	-0.627174	0.829057	0.674844	0.396705	0.020920	1.000000	nan	nan	nan	nan	nan	nan
ZIP Code	nan	nan	nan	nan	nan	nan	1.000000	-0.003720	-0.005687	-0.001452	-0.437979	-0.002896
Model Year	nan	nan	nan	nan	nan	nan	-0.003720	1.000000	-0.133244	-0.223901	0.017605	-0.113277
Electric Range	nan	nan	nan	nan	nan	nan	-0.005687	-0.133244	1.000000	0.058873	0.039889	0.038348
Base MSRP	nan	nan	nan	nan	nan	nan	-0.001452	-0.223901	0.058873	1.000000	0.006514	0.008548
Legislative District	nan	nan	nan	nan	nan	nan	-0.437979	0.017605	0.039889	0.006514	1.000000	-0.000326
DOL Vehicle ID	nan	nan	nan	nan	nan	nan	-0.002896	-0.113277	0.038348	0.008548	-0.000326	1.000000

Далее продолжим работу со столбцом `Brand`. Получим количество автомобилей определенного бренда:

```
data['Brand'].value_counts(ascending=False)
```

Результат:

```
TESLA        42299
NISSAN       12664
CHEVROLET     9433
FORD          5774
KIA           4797
             ...
LEXUS            1
MG               1
MINI             1
SONO             1
DODGE            1
Name: Brand, Length: 67, dtype: int64
```

Можем получить количество уникальных брендов:

```
data['Brand'].nunique()
```

Результат — 67 брендов.

Получаем список брендов:

```
Brands = data['Brand'].unique()
Brands
```

Результат:

```
array(['TESLA ', 'VOLKSWAGEN ', 'POLESTAR ', 'BMW ', 'HONDA ', 'LUCID ',
       'PEUGEOT ', 'AUDI ', 'MERCEDES ', 'NISSAN ', 'HYUNDAI ',
       'PORSCHE ', 'MG ', 'MINI ', 'OPEL ', 'SKODA ', 'VOLVO ', 'KIA ',
       'RENAULT ', 'MAZDA ', 'LEXUS ', 'CUPRA ', 'SEAT ', 'LIGHTYEAR ',
       'AIWAYS ', 'DS ', 'CITROEN ', 'JAGUAR ', 'FORD ', 'BYTON ',
       'SONO ', 'SMART ', 'FIAT ', 'NISSAN', 'BMW', 'TESLA', 'VOLKSWAGEN',
       'CHEVROLET', 'TOYOTA', 'KIA', 'FORD', 'CHRYSLER', 'AUDI',
       'MITSUBISHI', 'FISKER', 'HYUNDAI', 'FIAT', 'JEEP', 'SMART',
       'VOLVO', 'PORSCHE', 'CADILLAC', 'JAGUAR', 'HONDA', 'MERCEDES-BENZ',
       'MINI', 'LUCID MOTORS', 'POLESTAR', 'RIVIAN', 'LINCOLN',
       'LAND ROVER', 'SUBARU', 'AZURE DYNAMICS', 'WHEEGO ELECTRIC CARS',
       'TH!NK', 'BENTLEY', 'DODGE'], dtype=object)
```

Один из возможных способов категоризации — использование `get_dummies()`:

```
dds = pd.get_dummies(data, columns=['Brand'], drop_first= True )
```

При реализации таким образом мы получаем генерацию еще 67 признаков.

Выполним многоклассовую категоризацию. Для ее реализации необходимо подключить дополнительную библиотеку:

```
from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()
data_labeled = data.copy()
data_labeled.loc[:, 'Brand'] =
labelencoder.fit_transform(data_labeled.loc[:, 'Brand'])
data_labeled
```

Получаем в таблице дополнительный столбец, содержащий числовые значения. Это очень удобно. Данные столбца можно использовать для различного анализа, в том числе в машинном обучении. Фрагмент результата:

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km
0	59	Model 3 Long Range Dual Motor	4.6	233.0	450.0
1	63	ID.3 Pure	10.0	160.0	270.0
2	47	2	4.7	210.0	400.0
3	6	iX3	6.8	180.0	360.0
4	21	e	9.5	145.0	170.0

Построим еще раз корреляционную матрицу:

```
corr = data_labeled.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Результат:

	Brand	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro	ZIP Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID
Brand	1.000000	0.071535	0.083302	0.079599	-0.218089	0.086698	-0.049544	-0.003713	0.223068	0.209012	0.018670	0.058324	0.008834
AccelSec	0.071535	1.000000	-0.786195	-0.677062	-0.382904	-0.175335	-0.627174	nan	nan	nan	nan	nan	nan
TopSpeed_KmH	0.083302	-0.786195	1.000000	0.746662	0.355675	0.126470	0.829057	nan	nan	nan	nan	nan	nan
Range_Km	0.079599	-0.677062	0.746662	1.000000	0.313077	0.300163	0.674844	nan	nan	nan	nan	nan	nan
fficiency_WhKm	-0.218089	-0.382904	0.355675	0.313077	1.000000	0.301230	0.396705	nan	nan	nan	nan	nan	nan
Seats	0.086698	-0.175335	0.126470	0.300163	0.301230	1.000000	0.020920	nan	nan	nan	nan	nan	nan
PriceEuro	-0.049544	-0.627174	0.829057	0.674844	0.396705	0.020920	1.000000	nan	nan	nan	nan	nan	nan
ZIP Code	-0.003713	nan	nan	nan	nan	nan	nan	1.000000	-0.003720	-0.005687	-0.001452	-0.437979	-0.002896
Model Year	0.223068	nan	nan	nan	nan	nan	nan	-0.003720	1.000000	-0.133244	-0.223901	0.017605	-0.113277
Electric Range	0.209012	nan	nan	nan	nan	nan	nan	-0.005687	-0.133244	1.000000	0.058873	0.039889	0.038348
Base MSRP	0.018670	nan	nan	nan	nan	nan	nan	-0.001452	-0.223901	0.058873	1.000000	0.006514	0.008548
Legislative District	0.058324	nan	nan	nan	nan	nan	nan	-0.437979	0.017605	0.039889	0.006514	1.000000	-0.000326
DOL Vehicle ID	0.008834	nan	nan	nan	nan	nan	nan	-0.002896	-0.113277	0.038348	0.008548	-0.000326	1.000000

Видим, что нет высоких значений коэффициента корреляции по рассматриваемому столбцу. Но мы можем аналогичным образом поработать с другими столбцами таблицы.

Поработаем с присутствующими в матрице корреляции значениями `NaN`, заменив их средними значениями:

```
data_labeled['AccelSec'].fillna(data_labeled['AccelSec'].mean(),
inplace = True)

data_labeled['TopSpeed_KmH'].fillna(data_labeled['TopSpeed_KmH'].m
ean(), inplace = True)

data_labeled['Range_Km'].fillna(data_labeled['Range_Km'].mean(),
inplace = True)

data_labeled['Efficiency_WhKm'].fillna(data_labeled['Efficiency_Wh
Km'].mean(), inplace = True)

data_labeled['Seats'].fillna(data_labeled['Seats'].mean(), inplace
= True)

data_labeled['PriceEuro'].fillna(data_labeled['PriceEuro'].mean(),
inplace = True)

data_labeled['ZIP Code'].fillna(data_labeled['ZIP Code'].mean(),
inplace = True)

data_labeled['Model Year'].fillna(data_labeled['Model
Year'].mean(), inplace = True)
```

```
data_labeled['Electric Range'].fillna(data_labeled['Electric
Range'].mean(), inplace = True)

data_labeled['Base MSRP'].fillna(data_labeled['Base MSRP'].mean(),
inplace = True)

data_labeled['Legislative
District'].fillna(data_labeled['Legislative District'].mean(),
inplace = True)

data_labeled['DOL Vehicle ID'].fillna(data_labeled['DOL Vehicle
ID'].mean(), inplace = True)
```

Построим матрицу корреляции еще раз:

```
corr = data_labeled.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Результат:

	Brand	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro	ZIP Code	Model Year	Electric Range	Base MSRP	Legislative District	DOL Vehicle ID
Brand	1.000000	0.002306	0.002686	0.002566	-0.007031	0.002795	-0.001597	-0.003711	0.222947	0.208899	0.018660	0.058208	0.008829
AccelSec	0.002306	1.000000	-0.786195	-0.677062	-0.382904	-0.175335	-0.627174	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
TopSpeed_KmH	0.002686	-0.786195	1.000000	0.746662	0.355675	0.126470	0.829057	-0.000000	0.000000	-0.000000	0.000000	-0.000000	-0.000000
Range_Km	0.002566	-0.677062	0.746662	1.000000	0.313077	0.300163	0.674844	0.000000	-0.000000	0.000000	-0.000000	0.000000	0.000000
Efficiency_WhKm	-0.007031	-0.382904	0.355675	0.313077	1.000000	0.301230	0.396705	-0.000000	0.000000	-0.000000	0.000000	-0.000000	-0.000000
Seats	0.002795	-0.175335	0.126470	0.300163	0.301230	1.000000	0.020920	-0.000000	0.000000	-0.000000	0.000000	-0.000000	-0.000000
PriceEuro	-0.001597	-0.627174	0.829057	0.674844	0.396705	0.020920	1.000000	-0.000000	0.000000	-0.000000	0.000000	-0.000000	-0.000000
ZIP Code	-0.003711	0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	1.000000	-0.003720	-0.005687	-0.001452	-0.046935	-0.002896
Model Year	0.222947	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000	-0.003720	1.000000	-0.133244	-0.223901	0.017586	-0.113277
Electric Range	0.208899	0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.005687	-0.133244	1.000000	0.058873	0.039832	0.038348
Base MSRP	0.018660	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000	-0.001452	-0.223901	0.058873	1.000000	0.006505	0.008548
Legislative District	0.058208	0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.046935	0.017586	0.039832	0.006505	1.000000	-0.000326
DOL Vehicle ID	0.008829	0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.002896	-0.113277	0.038348	0.008548	-0.000326	1.000000

Видим, что вместо значений `NaN` в матрице появились нули. Это справедливо с точки зрения математики. Но по крайней мере, таблица выглядит более репрезентативно.

Дополнительные материалы для самостоятельного изучения

1. pandas.DataFrame.corr — pandas 1.5.3 documentation
2. Animation — Matplotlib 3.7.1 documentation

3. Plotly Python Graphing Library
4. Analyze Your Runkeeper Fitness Data | Chan`s Jupyter (goodboychan.github.io)
5. Cohort Analysis with Python | Aman Kharwal (thecleverprogrammer.com)

Для демонстрации примеров в п. 1–3 используется датасет Anime Recommendations Database.