

1 часть задачи. Для начала, загружу данные и проверю что датасет успешно загружен

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from pandas.plotting import scatter_matrix

d = pd.read_csv("dataset_2task_final.csv")
d.head()
```

```
Out[1]:
```

	landmarks_n	map_quality	rent_cost	route_rating
0	107.450712	12.224891	3.175834	116.521789
1	97.926035	13.715135	2.986853	104.642491
2	109.715328	7.688631	1.768171	106.745418
3	122.845448	12.268211	0.667700	124.682481
4	96.487699	8.523100	2.803592	100.959460

Посмотрю базовые описательные статистики

```
In [2]: d[["map_quality", "landmarks_n", "rent_cost", "route_rating"]].describe()
```

```
Out[2]:
```

	map_quality	landmarks_n	rent_cost	route_rating
count	500.000000	500.000000	500.000000	500.000000
mean	10.068781	100.102570	2.105407	106.246938
std	2.037073	14.718799	1.125671	12.531192
min	3.678514	51.380990	-0.948706	64.135154
25%	8.775920	89.495389	1.367799	98.226329
50%	10.042785	100.191957	2.120918	105.584578
75%	11.364996	109.551749	2.881225	115.205194
max	16.282461	157.790972	5.113061	147.227326

```
In [3]: v = ["map_quality", "landmarks_n", "rent_cost", "route_rating"]
```

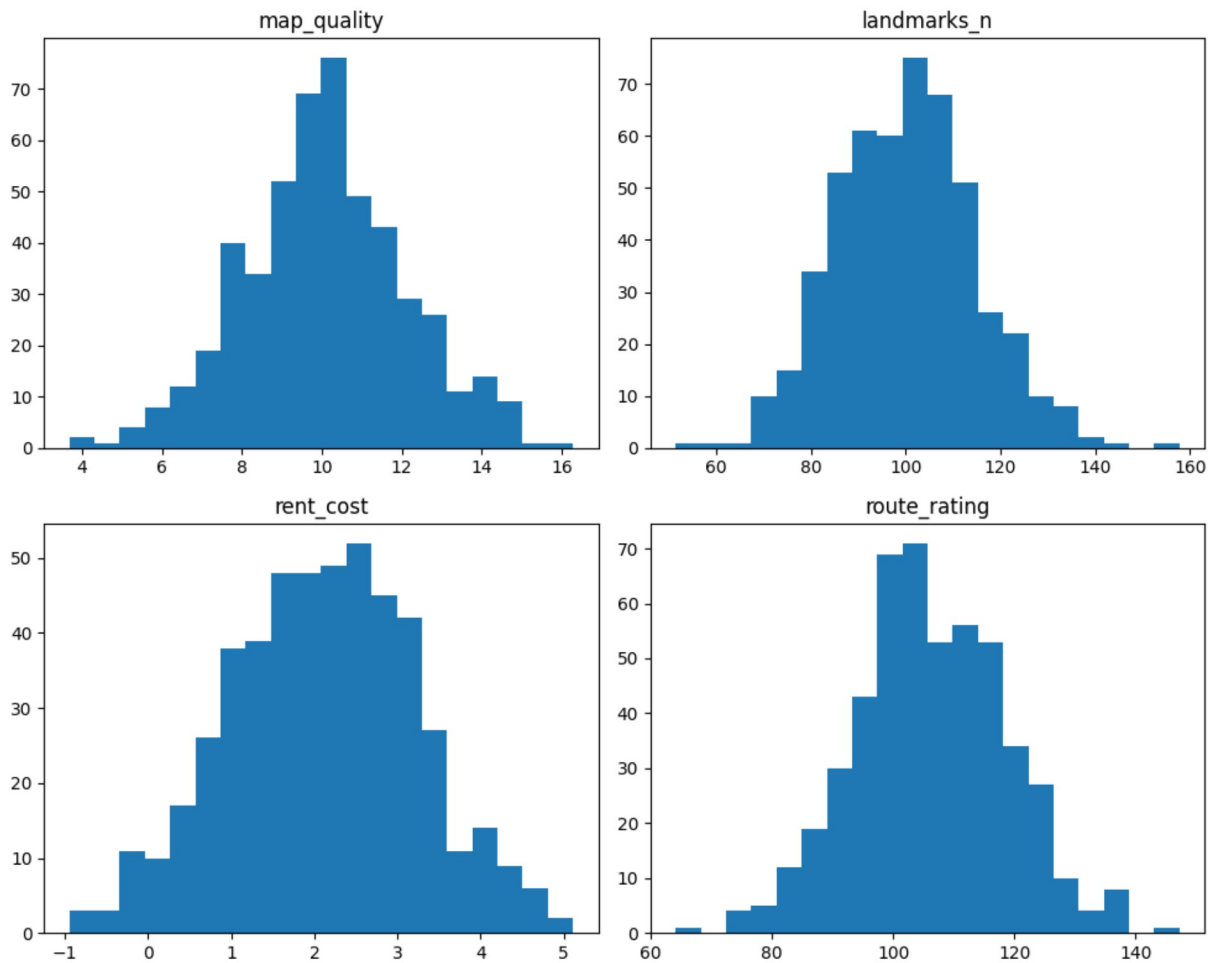
Составлю графики по всем столбцам

```
In [4]: fig, ax = plt.subplots(2, 2, figsize=(10, 8))
ax = ax.ravel()

for i, c in enumerate(v):
    ax[i].hist(d[c].dropna(), bins=20)
```

```
ax[i].set_title(c)

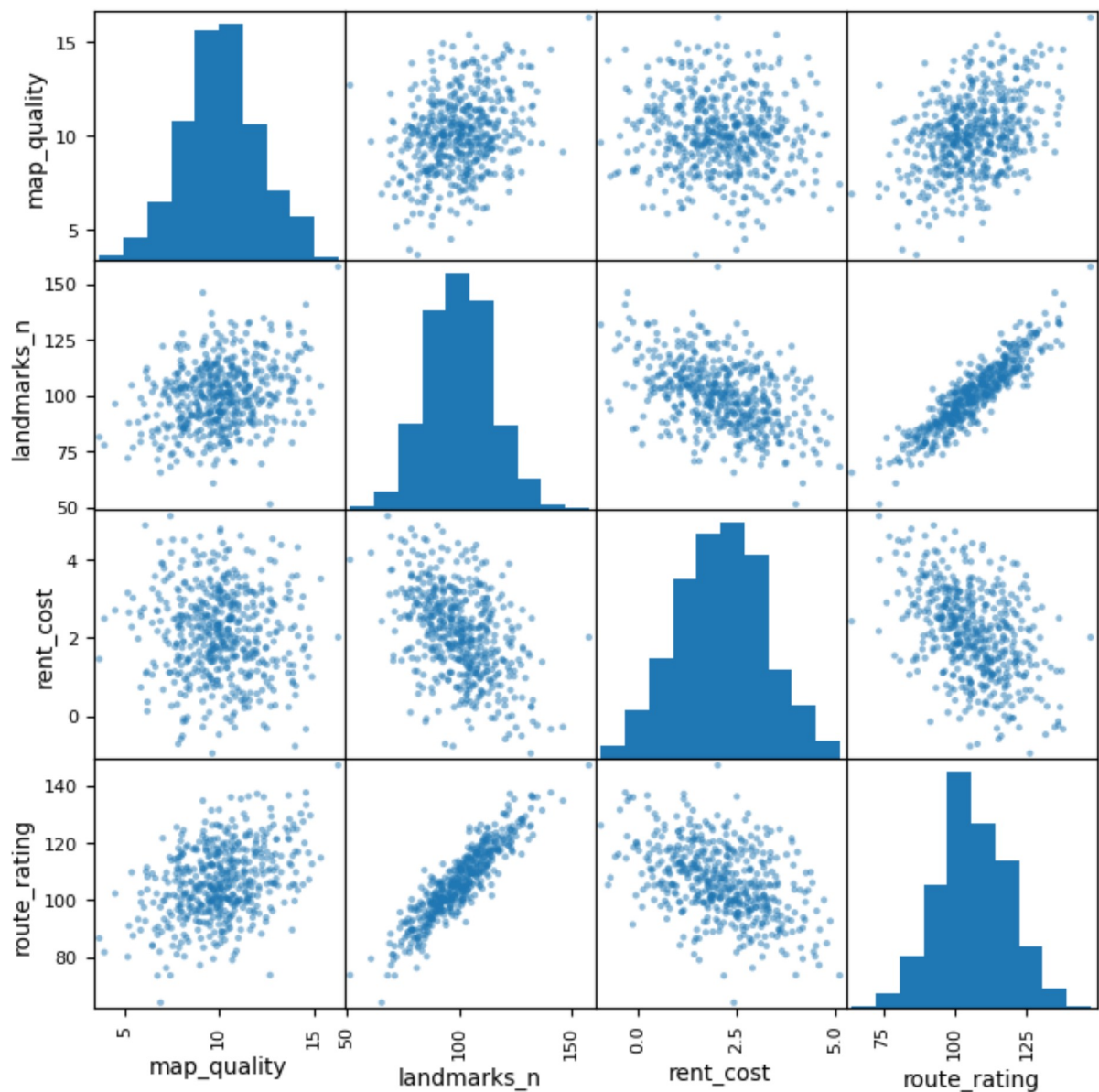
plt.tight_layout()
plt.show()
```



По map_quality Распределение близко к нормальному По landmarks_n тоже, заметна широкая дисперсия По rent_cost Распределение скошенное, есть маршруты с заметно более высокой стоимостью По route_rating распределение нормальное, достаточно компактное

Создам диаграммы попарных связей

```
In [5]: scatter_matrix(d[v], figsize=(8, 8))
plt.show()
```



По диаграммам рассеяния видно:

Между route_rating и landmarks_n - плотная почти линейная положительная связь.

Между route_rating и map_quality - тоже положительная линейная связь, но заметно слабее. Между route_rating и rent_cost - отрицательная связь: более дорогие маршруты, по-видимому, в среднем получают меньший рейтинг. Между landmarks_n и rent_cost - отрицательная связь: там, где много достопримечательностей, аренда, видимо, дешевле.

Посмотрю попарную корреляцию

```
In [6]: corr = d[v].corr()
corr
```

Out[6]:

	map_quality	landmarks_n	rent_cost	route_rating
map_quality	1.000000	0.288614	-0.066426	0.387550
landmarks_n	0.288614	1.000000	-0.444132	0.909559
rent_cost	-0.066426	-0.444132	1.000000	-0.437450
route_rating	0.387550	0.909559	-0.437450	1.000000

$\text{corr}(\text{route_rating}, \text{landmarks_n}) \sim 0.91$ - очень сильная положительная связь

$\text{corr}(\text{route_rating}, \text{map_quality}) \sim 0.39$ - умеренная положительная связь
 $\text{corr}(\text{route_rating}, \text{landmarks_n}) \sim -0.44$ - умеренная отрицательная связь

Ответ на первый вопрос

Сильнее всего влияют на рейтинг

В первую очередь, landmarks_n (количество достопримечательностей) - самая высокая корреляция (≈ 0.91) и по графикам видна сильная линейная связь

Потом rent_cost — заметная отрицательная корреляция с рейтингом (≈ -0.44)

Далее map_quality - умеренная положительная корреляция (≈ 0.39)

2. Оценка трех регрессионных моделей

```
In [7]: y = d["route_rating"]
```

Модель 1

```
In [12]: X1 = sm.add_constant(d["map_quality"])
m1 = sm.OLS(y, X1).fit()
t1 = pd.DataFrame({"coef": m1.params, "t": m1.tvalues, "p": m1.pvalues})
m1_rs2, m1_rs2a = m1.rsquared, m1.rsquared_adj
(t1, m1_rs2, m1_rs2a)
```

```
Out[12]: (
           coef      t      p
const    82.242584  31.506125  1.206558e-120
map_quality  2.384038   9.381716  2.293683e-19,
np.float64(0.15019472194638717),
np.float64(0.14848828564507466))
```

Результат

$b_0 = 82.24$, $t=31.51$, $p \sim 1.2 \cdot 10^{-120}$

$b_1 = 2.38$, $t=9.38$, $p = 2.3 \cdot 10^{-19}$

$R^2_{\text{squared}} = 0.150$

R1adj_squared = 0.148

Модель 2

```
In [13]: X2 = sm.add_constant(d[["map_quality", "landmarks_n"]])
m2 = sm.OLS(y, X2).fit()
t2 = pd.DataFrame({"coef": m2.params, "t": m2.tvalues, "p": m2.pvalues})
m2_rs2, m2_rs2a = m2.rsquared, m2.rsquared_adj
t2, m2_rs2, m2_rs2a
```

```
Out[13]: (
           coef          t          p
const      23.636690  14.125345  2.498070e-38
map_quality  0.839073   7.379636  6.716994e-13
landmarks_n  0.740858  47.079977  2.562799e-185,
np.float64(0.8443525169877154),
np.float64(0.8437261689675453))
```

b0 = 23.64, t=14.13, p=2.5*10^-38

b1 = 0.84, t=7.38, p = 6.7 * 10^-13

b2 = 0.74, t = 47.08, p = 2.6 * 10^-185

R2_squared = 0.844

R2_adj_squared = 0.844

Модель 3

```
In [14]: X3 = sm.add_constant(d[["map_quality", "landmarks_n", "rent_cost"]])
m3 = sm.OLS(y, X3).fit()
t3 = pd.DataFrame({"coef": m3.params, "t": m3.tvalues, "p": m3.pvalues})
m3_rs2, m3_rs2a = m3.rsquared, m3.rsquared_adj
(t3, m3_rs2, m3_rs2a)
```

```
Out[14]: (
           coef          t          p
const      26.721100  13.189709  2.899359e-34
map_quality  0.860822   7.597542  1.516092e-13
landmarks_n  0.720146  41.237816  2.170693e-162
rent_cost   -0.584217  -2.666333  7.918750e-03,
np.float64(0.8465519382375972),
np.float64(0.8456238249608085))
```

b0 = 26.72, t=13.19, p=2.9*10^-34

b1 = 0.86, t=7.60, p=1.5*10^-13

b2 = 0.72, t=41.24, p = 2.2*10^-162

b3 = -0.58, t=-2.67, p=0.0079

R3_squared = 0.847

R3_adj_squared = 0.846 Все коэффициенты значимы на уровне 1%

3. Соберу коэффы в одну таблицу

```
In [15]: pd.DataFrame({
    "model": ["M1", "M2", "M3"],
    "b0": [m1.params["const"], m2.params["const"], m3.params["const"]],
    "b1_map": [m1.params["map_quality"], m2.params["map_quality"], m3.params["map_q
  })
```

```
Out[15]:
```

	model	b0	b1_map
0	M1	82.242584	2.384038
1	M2	23.636690	0.839073
2	M3	26.721100	0.860822

Модель 1 учитывает только Прометку. Поскольку Прометка положительно коррелирует с числом достопримечательностей, а достопримечательности сами сильно повышают рейтинг, то часть эффекта Достопримечательностей завешивается на Прометку. Поэтому b1 в M1 завышен (2.38)

В Модели 2 мы вводим landmarks_n. Теперь модель отдельно учитывает вклад количества достопримечательностей, и чистый эффект Прометки оказывается гораздо меньше: b1 падает до 0.84

При переходе от M2 к M3 мы добавляем rent_cost. Прометка почти не коррелирует со стоимостью аренды, поэтому b1 меняется мало (0.84 → 0.86).

Константа b0 меняется, поскольку она - значение рейтинга при нулевых значениях всех объясняющих. Когда мы добавляем новые переменные и по-другому разлагаем среднее значение рейтинга на части, точка пересечения с осью - константа перенастраивается

4 задание

```
In [16]: [m1_rs2, m2_rs2, m3_rs2], [m1_rs2a, m2_rs2a, m3_rs2a]
```

```
Out[16]: ([np.float64(0.15019472194638717),
    np.float64(0.8443525169877154),
    np.float64(0.8465519382375972)],
  [np.float64(0.14848828564507466),
    np.float64(0.8437261689675453),
    np.float64(0.8456238249608085)])
```

Добавление переменной никогда не уменьшает обычный R^2 , модель всегда может хотя бы воспроизвести старое решение (поставив новый коэффициент равным нулю), а может и улучшить подгонку.

Переход от M1 к M2 даёт резкий скачок R^2 с 0.15 до 0.84, потому что landmarks_n чрезвычайно сильно связан с рейтингом.

Переход от M2 к M3 даёт небольшое улучшение R^2 (0.844 → 0.847): стоимость аренды объясняет дополнительную, но небольшую долю вариации рейтинга.

5. Стандартизирую переменные

```
In [17]: for c in ["map_quality", "landmarks_n", "rent_cost"]:
         d[c + "_z"] = (d[c] - d[c].mean()) / d[c].std()
```

Модель 1Z, рейтинг - прометка Z

```
In [18]: X1z = sm.add_constant(d["map_quality_z"])
         m1z = sm.OLS(y, X1z).fit()
         t1z = pd.DataFrame({"coef": m1z.params, "t": m1z.tvalues, "p": m1z.pvalues})
         m1z_rs2, m1z_rs2a = m1z.rsquared, m1z.rsquared_adj
         (t1z, m1z_rs2, m1z_rs2a)
```

```
Out[18]: (
           coef      t      p
const      106.246938  205.453581  0.000000e+00
map_quality_z  4.856459   9.381716  2.293683e-19,
np.float64(0.15019472194638717),
np.float64(0.14848828564507466))
```

Модель 2Z

```
In [20]: X2z = sm.add_constant(d[["map_quality_z", "landmarks_n_z"]])
         m2z = sm.OLS(y, X2z).fit()
         t2z = pd.DataFrame({"coef": m2z.params, "t": m2z.tvalues, "p": m2z.pvalues})
         m2z_rs2, m2z_rs2a = m2z.rsquared, m2z.rsquared_adj
         (t2z, m2z_rs2, m2z_rs2a)
```

```
Out[20]: (
           coef      t      p
const      106.246938  479.585339  0.000000e+00
map_quality_z  1.709252   7.379636  6.716994e-13
landmarks_n_z  10.904543  47.079977  2.562799e-185,
np.float64(0.8443525169877154),
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x401c4080>)
```

Модель 3Z

```
In [21]: X3z = sm.add_constant(d[["map_quality_z", "landmarks_n_z", "rent_cost_z"]])
         m3z = sm.OLS(y, X3z).fit()
         t3z = pd.DataFrame({"coef": m3z.params, "t": m3z.tvalues, "p": m3z.pvalues})
         m3z_rs2, m3z_rs2a = m3z.rsquared, m3z.rsquared_adj
         (t3z, m3z_rs2, m3z_rs2a)
```

```
Out[21]: (
           coef      t      p
const      106.246938  482.523967  0.000000e+00
map_quality_z  1.753556   7.597542  1.516092e-13
landmarks_n_z  10.599679  41.237816  2.170693e-162
rent_cost_z   -0.657636  -2.666333  7.918750e-03,
np.float64(0.8465519382375972),
np.float64(0.8456238249608085))
```

6. Как изменилась интерпретация b0, b1

До:

- b0 - прогноз рейтинга, когда все объясняющие переменные равны 0 в исходных единицах (0 баллов прометки, 0 достопримечательностей, 0 тыс. руб. аренды). В наших данных такие значения не встречаются, поэтому интерпретация скорее формальная
- b1 - изменение рейтинга при увеличении Прометки на 1, когда все остальное фиксировано

После:

- b0 во всех трёх стандартизированных моделях \approx среднему рейтингу маршрутов, потому что стандартизированные признаки имеют среднее 0. То есть теперь b0 - ожидаемый рейтинг маршрута при среднем качестве прометки, среднем числе достопримечательностей и средней стоимости аренды
- b1 теперь показывает, на сколько в среднем изменится рейтинг, если Прометка увеличивается на 1 стандартное отклонение (то есть существенно улучшается), при фиксированных остальных переменных

```
In [22]: [m1_rs2, m2_rs2, m3_rs2], [m1z_rs2, m2z_rs2, m3z_rs2]
```

```
Out[22]: ([np.float64(0.15019472194638717),
           np.float64(0.8443525169877154),
           np.float64(0.8465519382375972)],
          [np.float64(0.15019472194638717),
           np.float64(0.8443525169877154),
           np.float64(0.8465519382375972)])
```

7. R2 совпадают для стандартизированных и нет данных. Это происходит, потому что стандартизация - линейное преобразование, которое не изменяет ни линейную оболочку признаков, ни форму линейной комбинации в пространстве y. То есть "подпространство", натянутое на объясняющие переменные, остаётся тем же, а значит, те же самые прогнозы y_{pred} достигаются другими коэфами. Поскольку R2 зависит только от приближенности y_{pred} к y, он сохранится

8. Сравню оценку влияния прометки до и после добавления landmarks_n

Без них было 2.38

С ними стало 0.84

Эффект Прометки существенно уменьшается, но остаётся положительным и значимым.

Теоретический механизм:

Прометка положительно коррелирует с числом достопримечательностей: на маршрутах с большим количеством интересных объектов муниципалитет/операторы, скорее всего, больше инвестируют в навигацию.

Достопримечательности сами по себе сильно повышают рейтинг (большой коэффициент и огромная t-статистика).

Если не учитывать число достопримечательностей, модель "переписывает" на Прометку часть влияния достопримечательностей. Поэтому в M1 влияние Прометки завышено.

После добавления landmarks_n модель разделяет эффекты: большая часть положительного влияния на рейтинг приписывается достопримечательностям, а у Прометки остаётся чистый эффект

Это смещение происходит из-за пропущенной переменной - промека и достопримечательности сильно коррелируют, и пропуск значим

9. Сгенерирую 100 случайных признаков

```
In [23]: np.random.seed(0)
r = []
for i in range(100):
    c = f"r{i+1}"
    d[c] = np.random.randn(len(d))
    r.append(c)
```

C:\Users\dan\AppData\Local\Temp\ipykernel_9256\841537846.py:5: PerformanceWarning: Dataframe is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using `pd.concat(axis=1)` instead. To get a de-fragmented frame, use `newframe = frame.copy()`

```
d[c] = np.random.randn(len(d))
```

C:\Users\dan\AppData\Local\Temp\ipykernel_9256\841537846.py:5: PerformanceWarning: Dataframe is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using `pd.concat(axis=1)` instead. To get a de-fragmented frame, use `newframe = frame.copy()`

```
d[c] = np.random.randn(len(d))
```

C:\Users\dan\AppData\Local\Temp\ipykernel_9256\841537846.py:5: PerformanceWarning: Dataframe is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using `pd.concat(axis=1)` instead. To get a de-fragmented frame, use `newframe = frame.copy()`

```
d[c] = np.random.randn(len(d))
```

C:\Users\dan\AppData\Local\Temp\ipykernel_9256\841537846.py:5: PerformanceWarning: Dataframe is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using `pd.concat(axis=1)` instead. To get a de-fragmented frame, use `newframe = frame.copy()`

```
d[c] = np.random.randn(len(d))
```

Сама модель

```
In [24]: Xb = sm.add_constant(d[["map_quality", "landmarks_n", "rent_cost"] + r])
mb = sm.OLS(y, Xb).fit()
mb_rs2, mb_rs2a = mb.rsquared, mb.rsquared_adj
mb_rs2, mb_rs2a
```

```
Out[24]: (np.float64(0.879722130037742), np.float64(0.8484377345677607))
```

R2 Слегка возросли

Это доказывает, что модель в целом правильно работает