Программная документация

Загружаем библиотеки и наши данные

```
df_2019.duplicated().sum()
```

```
np.int64(0)
```

```
df_2019 = df_2019.drop("Unnamed: 0", axis=1)
```

[124]

```
columns_for_under_zero = ['trip_distance', 'fare_amount', 'extra', 'tip_amount', 'tolls_amount',
       'total_amount', 'cluster']
for column in columns_for_under_zero:
    was_lines = len(df_2019)
    print("Убрано отрицательных строк у колонки",column , " : ", was_lines - len(df_2019.where(df_2019[column] > 0).dropna().reset_index().drop("index", axis=1)))
```

[137]

```
Убрано отрицательных строк у колонки trip_distance  :  0
Убрано отрицательных строк у колонки fare_amount  :  0
Убрано отрицательных строк у колонки extra  :  0
Убрано отрицательных строк у колонки tip_amount  :  0
Убрано отрицательных строк у колонки tolls_amount  :  0
Убрано отрицательных строк у колонки total_amount  :  0
Убрано отрицательных строк у колонки cluster  :  0
```

Проверяем данные на дубликаты и смотрим количество нулей, в наших ничего не нашлось

```
def work_with_proportion(df, year):
    df["tpep_pickup_datetime"] = pd.to_datetime(df["tpep_pickup_datetime"], format="%m/%d/%Y %I:%M:%S %p")
    df["tpep_dropoff_datetime"] = pd.to_datetime(df["tpep_dropoff_datetime"], format="%m/%d/%Y %I:%M:%S %p")

    df = df[df["tpep_dropoff_datetime"].apply(lambda x: x.year) == year]

    month_year = df.tpep_dropoff_datetime.apply(lambda x: x.month).unique()
    count_in_month = [df[df.tpep_dropoff_datetime.apply(lambda x: x.month) == month]["VendorID"].count() for month in month_year]
    count_in_month = list(map(lambda x: x//2 ,count_in_month))
    return pd.concat([df[df.tpep_dropoff_datetime.apply(lambda x: x.month) == month].head(count) for month, count in zip(month_year, count_in_month)]).reset_index(drop=True)
```

Делаем функцию для сохранения пропорции
Она смотрит оригинальную пропорцию в данных и убирает половину данных оставляя полученную пропорцию
Между тем убирает года которые не относятся к текущей таблице

```
df_2019["time_trip"] = (df_2019["tpep_dropoff_datetime"] - df_2019["tpep_pickup_datetime"]).apply(lambda x: x.total_seconds())
```

[126]

```
df_2019 = df_2019.drop(["tpep_dropoff_datetime","tpep_pickup_datetime"], axis=1)
```

[127]

Делаем столбец время пути, чтобы мы могли сделать кластеризацию, так как данные даты нельзя применить в модели

```
from sklearn.preprocessing import StandardScaler, Normalizer
```

[48]

```
scal = StandardScaler()
norm = Normalizer()
df_for_pca_2019 = pd.DataFrame(norm.fit_transform(scal.fit_transform(df_2019)), columns=df_2019.columns, index=df_2019.index)
```

[195]

```
from sklearn.decomposition import PCA
```

[58]

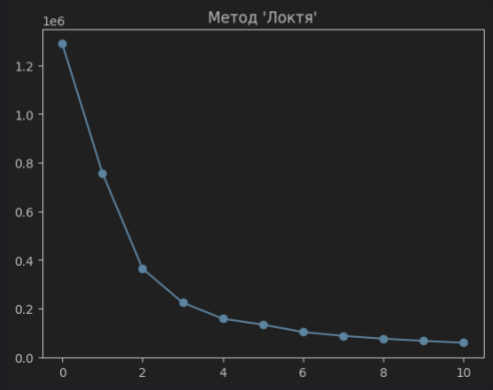
```
pca = PCA(n_components=2)
for_cluster_2019 = pca.fit_transform(df_for_pca_2019)
```

[196]

Масштабируем и нормализуем данные, чтобы на графике они были ближе друг к другу
И используем уменьшение размерности, чтобы перенести все наши колонки к двум колонкам

```
list_for_elbow = []
for n_clusters in range(1,12):
    kmeans = KMeans(n_clusters=n_clusters, random_state=0, init='k-means++')
    kmeans.fit(for_cluster_2019)
    list_for_elbow.append(kmeans.inertia_)
plt.title("Метод 'Локтя'")
plt.plot(list_for_elbow, marker='o')
[197]
```

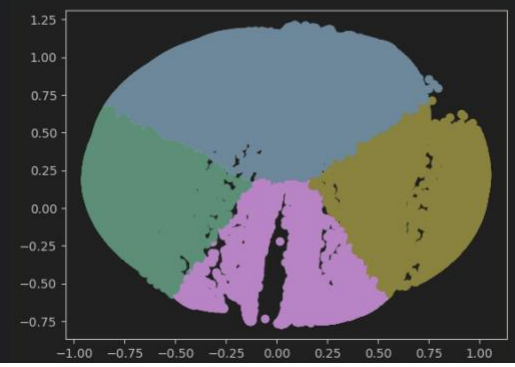
```
[<matplotlib.lines.Line2D at 0x13311b770>]
```



Данный график показывает оптимальное количество кластеров, определяется они просмотром изменения прошлого и следующего пункта в нашем случае не значительно изменилось от 4 до 5 и так далее, тогда берем 4 кластера

```
kmean = KMeans(n_clusters=n_cluster)
kmean.fit(for_cluster_2019)
labels = kmean.labels_
plt.scatter(for_cluster_2019[:, 0], for_cluster_2019[:, 1], c=labels, cmap='viridis')
[199]
```

```
<matplotlib.collections.PathCollection at 0x1429b0bc0>
```



Применяем модель для кластеризации по выбранным данным, для просмотра и отбора данных для последующих действий

```
columns_for_drop = ["VendorID", "passenger_count", "PULocationID", "DOLocationID", "mta_tax", "congestion_surcharge", "payment_type", "RatecodeID", "improvement_surcharge",
 "tolls_amount"]
[207]
```

После просмотра данных в каждом кластере удалим эти колонки они не изменяются среди кластеров

```
columns_for_under_zero = ['trip_distance', 'fare_amount', 'extra', 'tip_amount', 'tolls_amount',
       'total_amount', 'time_trip']
for column in columns_for_under_zero:
    was_lines = len(df_2019)
    df_2019 = df_2019[df_2019[column] >= 0]
    print("Убрано отрицательных строк у колонки",column , " : ", was_lines - len(df_2019))
[158]
```

```
Убрано отрицательных строк у колонки trip_distance  :  0
Убрано отрицательных строк у колонки fare_amount  :  5340
Убрано отрицательных строк у колонки extra  :  2
Убрано отрицательных строк у колонки tip_amount  :  0
Убрано отрицательных строк у колонки tolls_amount  :  0
Убрано отрицательных строк у колонки total_amount  :  0
Убрано отрицательных строк у колонки time_trip  :  972
```

Далее удаляем отрицательные значения в колонках, в которых это не может быть

```python
def full_update_df(df, year):
    df = df.drop("Unnamed: 0", axis=1)
    df = work_with_proportion(df, year)
    df["time_trip"] = (df["tpep_dropoff_datetime"] - df["tpep_pickup_datetime"]).apply(lambda x: x.total_seconds())
    df = df.drop(["tpep_dropoff_datetime","tpep_pickup_datetime"], axis=1)
    df = df.drop("store_and_fwd_flag",axis=1)

    columns_for_drop = ["VendorID", "passenger_count", "PULocationID", "DOLocationID", "mta_tax", "congestion_surcharge", "payment_type", "RatecodeID", "improvement_surcharge",
     "tolls_amount"]
    df = df.drop(columns_for_drop, axis=1)

    columns_for_under_zero = ['trip_distance', 'fare_amount', 'extra', 'tip_amount', 'tolls_amount',
       'total_amount', 'time_trip']
    for column in columns_for_under_zero:
        was_lines = len(df)
        df = df[df[column] >= 0]
        print("Убрано отрицательных строк у колонки",column , " : ", was_lines - len(df))
    return df
[166]
```

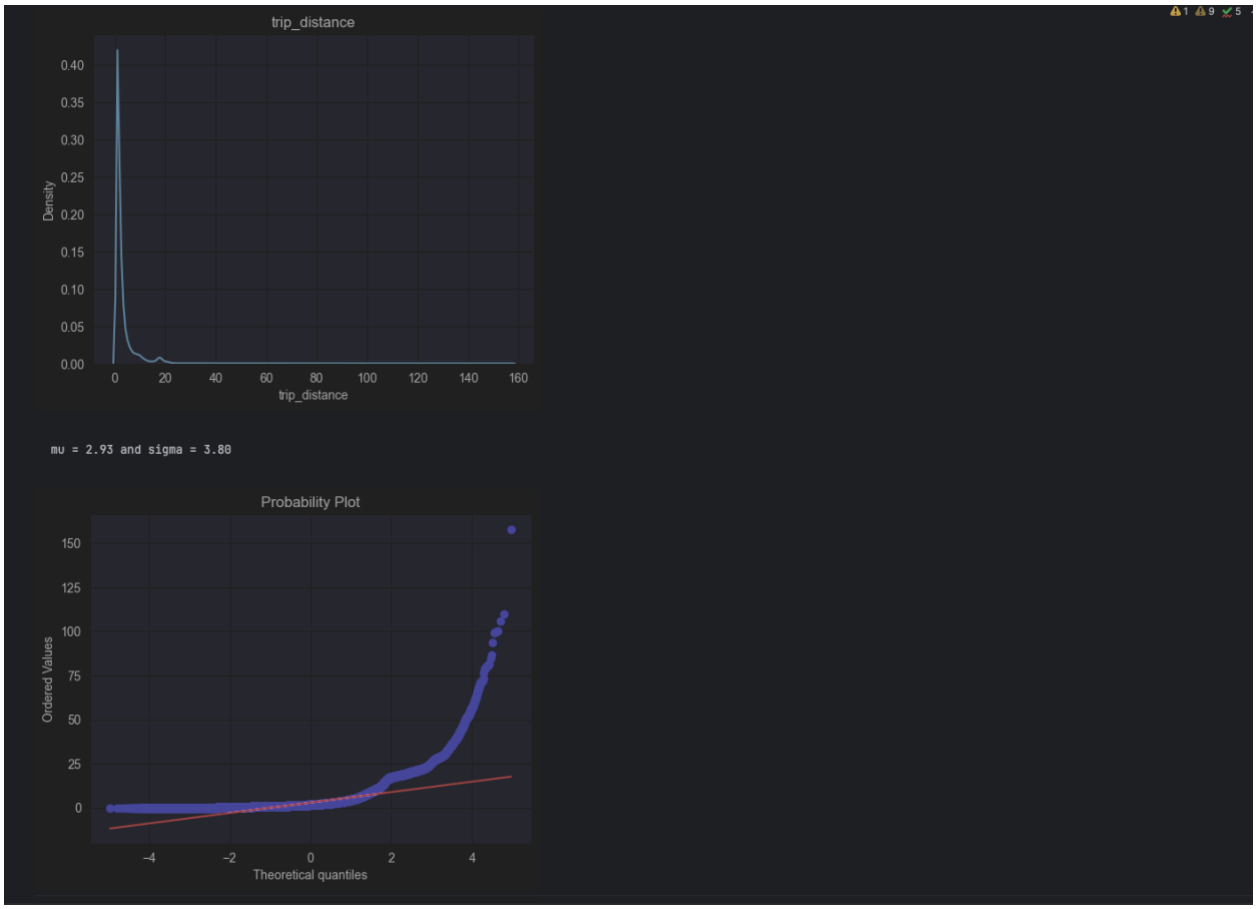
```python
df_2019 = full_update_df(df_2019, 2019)
[155]
```

```python
df_2020 = full_update_df(pd.read_csv("data_for_wr2020.csv"), 2020)
[167]
```

```
Убрано отрицательных строк у колонки trip_distance  :  13
Убрано отрицательных строк у колонки fare_amount  :  6389
Убрано отрицательных строк у колонки extra  :  0
Убрано отрицательных строк у колонки tip_amount  :  0
Убрано отрицательных строк у колонки tolls_amount  :  0
Убрано отрицательных строк у колонки total_amount  :  2
Убрано отрицательных строк у колонки time_trip  :  1
```

```python
df_2021 = full_update_df(pd.read_csv("data_for_wr2021.csv"), 2021)
[168]
```

Изменяем остальные таблицы, также как и первую



Смотрим плотность и нормальность данных в таблице

Далее мини задания

Районы, из которых чаще всего производилась посадка пассажиров,где чаевых было оставлено не менее 15% от суммы оплаты за поездку (длина поездки не должна превышать 2 км);

```
def task_1(df):
    print((((df.where((df["tip_amount"] / df["total_amount"]) >= 0,15)).where(df["trip_distance"] <= 2)).dropna()).groupby("PULocationID")["DOLocationID"].count().reset_index()
     .rename({"DOLocationID":"count_trips"}, axis=1).sort_values(by='count_trips', ascending=False).head())
```

✓ [256] 10ms

```
task_1(df_2019_for_tasks)
```

✓ [257] 360ms

	PULocationID	count_trips
219	237.0	70929
218	236.0	58358
147	161.0	53191
170	186.0	45352
148	162.0	45209

```
task_1(df_2020_for_tasks)
```

✓ [258] 285ms

	PULocationID	count_trips
218	237.0	70191
217	236.0	61195
147	161.0	55905
170	186.0	47080
148	162.0	46307

```
task_1(df_2021_for_tasks)
```

✓ [259] 240ms

	PULocationID	count_trips
224	237.0	80719
223	236.0	78054
132	141.0	45579
226	239.0	42638
152	161.0	41865

В данном модуле надо определить «Районы, из которых чаще всего производилась посадка пассажиров,где чаевых было оставлено не менее 15% от суммы оплаты за поездку (длина поездки не должна превышать 2 км);»
Можно увидеть, что главные локации в топе, с каждым годом растут по популярности в высадке по условию

Стоимость на километр поездки не превышала среднюю стоимость всех поездок на километр по такому же тарифу;

```
task_2(df_2019_for_tasks.copy())
```

✓ [361] 1s 221ms

```
В тарифе 1 кол-во не привышающих среднюю стоимость 1204464
В тарифе 2 кол-во не привышающих среднюю стоимость 48178
В тарифе 3 кол-во не привышающих среднюю стоимость 4476
В тарифе 4 кол-во не привышающих среднюю стоимость 976
В тарифе 5 кол-во не привышающих среднюю стоимость 4673
В тарифе 6 кол-во не привышающих среднюю стоимость 6
```

```
task_2(df_2020_for_tasks.copy())
```

✓ [362] 1s 381ms

```
В тарифе 1 кол-во не привышающих среднюю стоимость 1196729
В тарифе 2 кол-во не привышающих среднюю стоимость 50877
В тарифе 3 кол-во не привышающих среднюю стоимость 4179
В тарифе 4 кол-во не привышающих среднюю стоимость 1189
В тарифе 5 кол-во не привышающих среднюю стоимость 4715
В тарифе 6 кол-во не привышающих среднюю стоимость 5
```

```
task_2(df_2021_for_tasks.copy())
```

✓ [363] 1s 252ms

```
В тарифе 1 кол-во не привышающих среднюю стоимость 1193668
В тарифе 2 кол-во не привышающих среднюю стоимость 17670
В тарифе 3 кол-во не привышающих среднюю стоимость 1067
В тарифе 4 кол-во не привышающих среднюю стоимость 1015
В тарифе 5 кол-во не привышающих среднюю стоимость 2658
В тарифе 6 кол-во не привышающих среднюю стоимость 2
```

Среднее количество пассажиров на поездку, пользующихся услугами такси с самыми популярными тарифами.

```
task_3(df_2019_for_tasks)
✓ [321] 88ms
```

	RatecodeID	passenger_count_mean	Popularity
0	1	1.548060	1913359
1	2	1.584823	49916
2	3	1.631510	4608
3	4	1.509876	1367
4	5	1.458663	5564
5	6	0.875000	8

```
task_3(df_2020_for_tasks)
✓ [322] 97ms
```

	RatecodeID	passenger_count_mean	Popularity
0	1.0	1.536614	1916376
1	2.0	1.640970	54781
2	3.0	1.695643	4544
3	4.0	1.552439	1640
4	5.0	1.550552	11770
5	6.0	0.947368	19

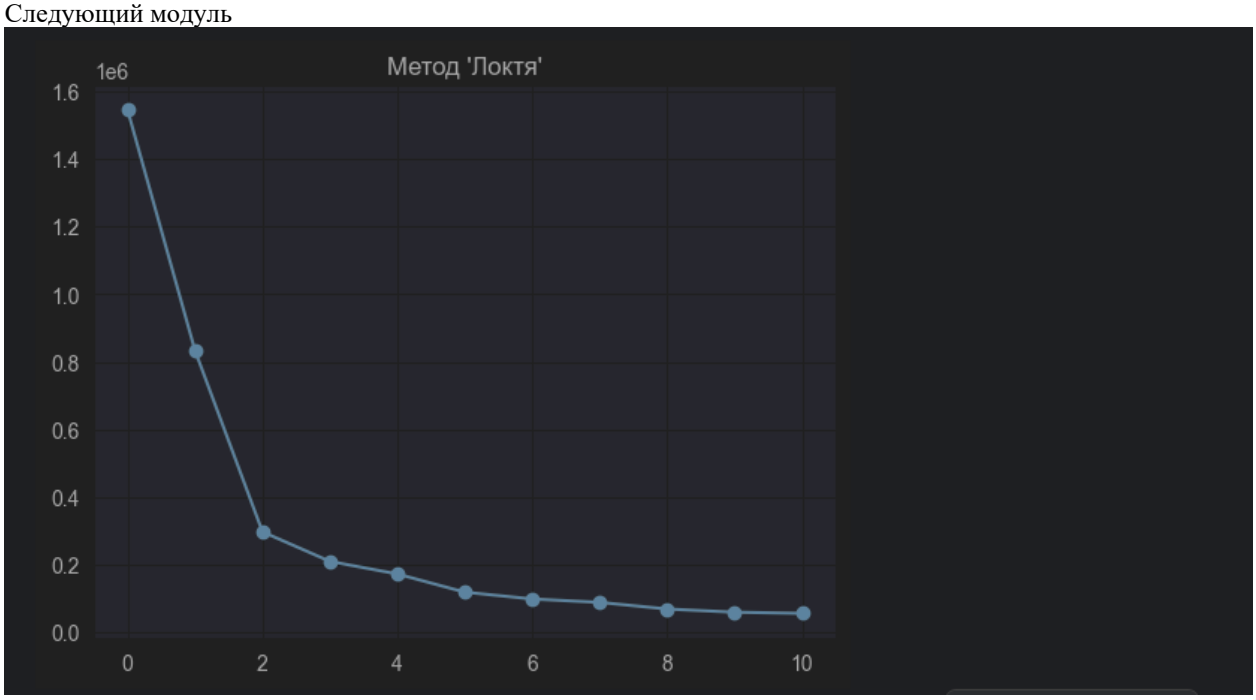
```
task_3(df_2021_for_tasks)
✓ [323] 81ms
```

	RatecodeID	passenger_count_mean	Popularity
0	1.0	1.406091	1862439
1	2.0	1.486674	19286
2	3.0	1.546099	1269
3	4.0	1.424422	1687
4	5.0	1.229506	7002
5	6.0	1.538462	13

Среднее количество пассажиров на поездку, пользующихся услугами такси с самыми популярными тарифами.

Популярность последнего года упала по отношению к предыдущим и количество людей уменьшилось на всех тарифах, кроме последнего тарифа

Следующий модуль

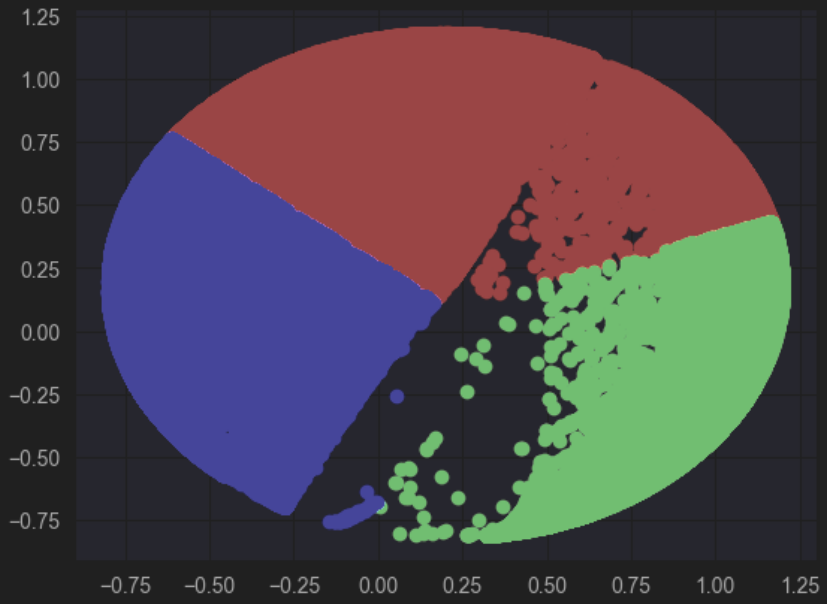


После обработки данных, количество кластеров будем использовать 3

KMeans

```
kmeans_show(df_2019.copy())      df_2019
✓ [30] 5s 218ms

0.5133657873611955
Кластер  0
Цвет  red
trip_distance  max:  157.8  min:  0.0  mean:  7.55
fare_amount  max:  756.0  min:  0.0  mean:  28.56
extra  max:  9.5  min:  0.0  mean:  0.72
tip_amount  max:  400.0  min:  0.0  mean:  4.91
total_amount  max:  766.8  min:  0.0  mean:  38.34
time_trip  max:  86392.0  min:  0.0  mean:  2818.65
Кластер  1
Цвет  green
trip_distance  max:  18.7  min:  0.0  mean:  1.73
fare_amount  max:  52.0  min:  0.0  mean:  9.75
```

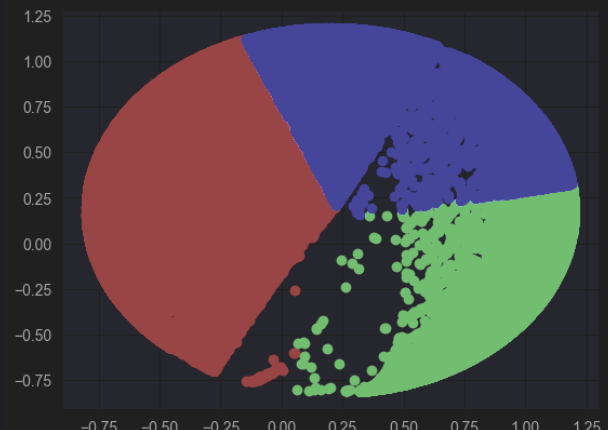


Рассмотрим кластеризатор KMEANS по индексу Дэвиса-Болдуина (Она вычисляет компактность как расстояние от объектов кластера до их центроидов, а отделимость - как расстояние между центроидами) Оценка ближе к нулю, что означает лучшее значение

Birch

```
Birch_show(df_2019.copy())
[25]
```

```
0.5007215088292014
Кластер  0
Цвет  red
trip_distance  max:  22.3  min:  0.0  mean:  1.63
fare_amount  max:  35.0  min:  0.0  mean:  9.16
extra  max:  1.3  min:  0.0  mean:  0.26
tip_amount  max:  15.0  min:  0.0  mean:  1.54
total_amount  max:  54.4  min:  0.0  mean:  14.07
time_trip  max:  86392.0  min:  0.0  mean:  798.63
Кластер  1
Цвет  green
trip_distance  max:  18.7  min:  0.0  mean:  1.67
fare_amount  max:  52.0  min:  0.0  mean:  9.54
```

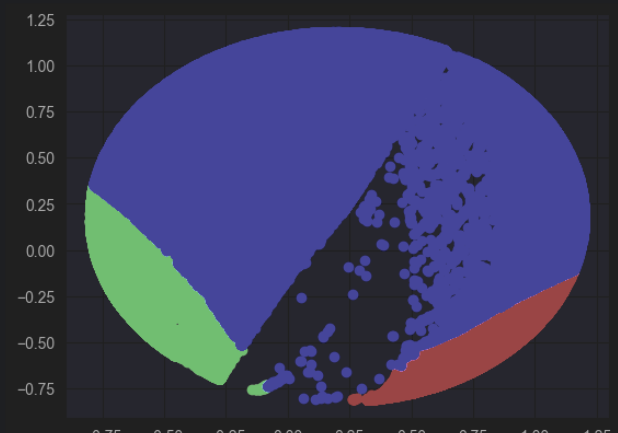


Далее кластеризатор Birch, по оценке он выше, на данном годе

GausianMixture

```
gaussian_mixture_show(df_2019.copy())
✓ [20] 29s 306ms
```

```
0.7008391181392156
Кластер  0
Цвет  red
trip_distance  max:  6.0  min:  0.0  mean:  1.39
fare_amount  max:  52.0  min:  0.0  mean:  8.4
extra  max:  87.56  min:  1.3  mean:  2.78
tip_amount  max:  5.0  min:  0.0  mean:  1.45
total_amount  max:  140.36  min:  2.5  mean:  13.43
time_trip  max:  4114.0  min:  0.0  mean:  686.17
Кластер  1
Цвет  green
trip_distance  max:  5.8  min:  0.0  mean:  1.22
fare_amount  max:  17.0  min:  0.0  mean:  7.42
```



Последний в рассмотрении GausianMixture оценка худшая из всех

Будем выбирать kmean
Кластер 0 под красным цветом распределны длинные поездки со средним количеством доплаты за поездку

Кластер 1 средняя дистанция поездки большой объем доплаты

Кластер 2 маленькая дистанция поездок с минимальным количеством доплаты

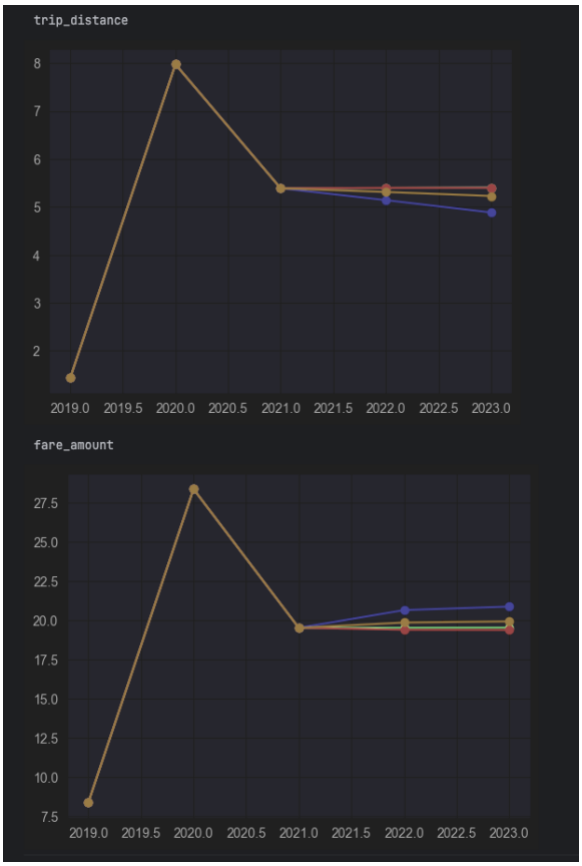
*Остальные колонки счетов и временем свзаны с дистанцией поездки

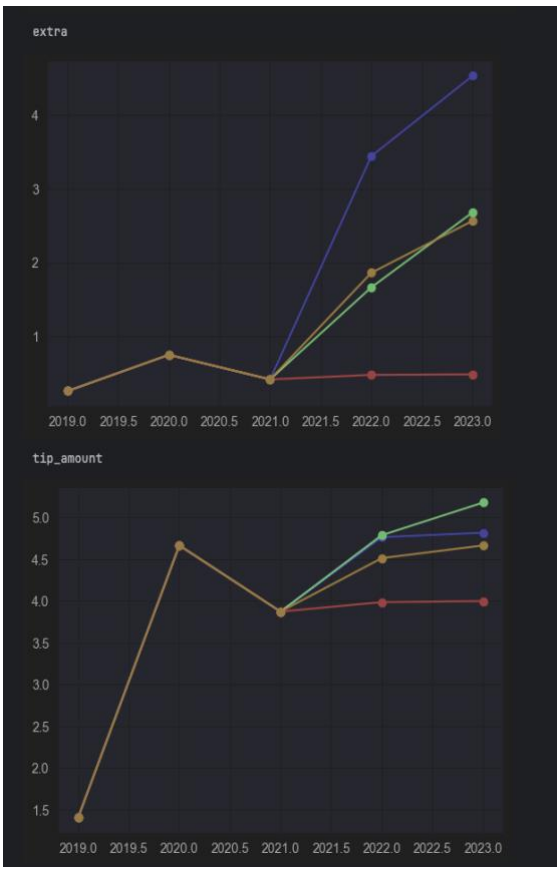
Модуль 3

Делаем средние данные по кластеру в каждом бою

Для предсказывания взял три модели

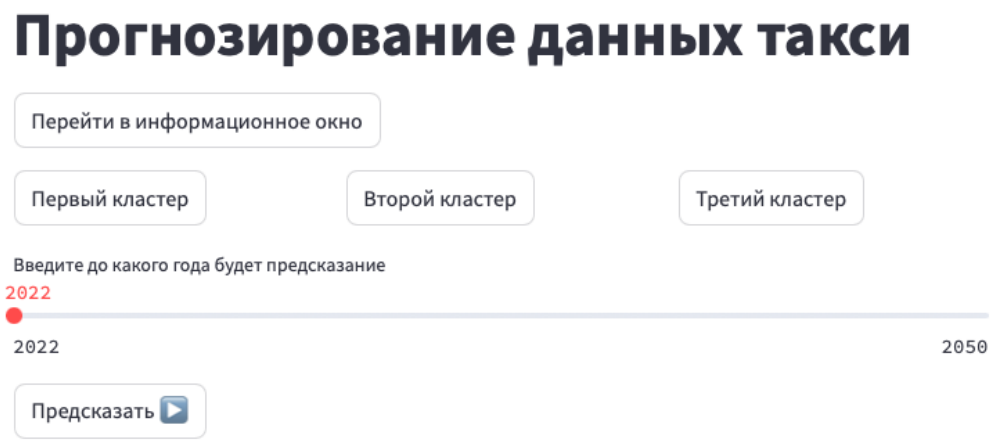
- GradientBoostingRegressor зеленый
- LinearRegression красный
- RandomForestRegression синий
- Среднее значение оранжевый





Для последующих действий взял RandomForestClassifier

Модуль приложения
Для графического интерфейса была взята библиотека streamlit



Графический интерфейс, для остальной информации по приложению следуюет посмотреть руководство пользователя