گزارش تمرین اول داده‌کاوی

علیرضا آزادبخت ۹۹۴۲۲۰۱۹

مقدمه:

در این تمرین دیتاستی شامل ۲۶۸۸۰۵۰ داده با بردار ویژگی ۴۹ از مشخصات خانه های اجاره ای در شهر های مختلف المان در اختیار ما قرار گرفته و در مورد این دیتاست باید به ۶ سوال مختلف پاسخ میدادیم، برای این کار ابتدا مراحل تمیز سازی داده ها را انجام دادیم و طی آن مقادیر null و داده های پرت رسیدگی کردیم و سپس مقداری EDA بر روی دیتاست انجام دادیم که تعدادی از سوالات را پاسخ دهیم، و سپس به پیاده سازی یک مدل رگرسیون از پایه پرداخته و نتایج مدل ساخته شده و مدل های پکیج های معروف را در شرایط گوناگون بر روی دیتاست محاسبه کردیم، برای این کار ویژگی livingSpace را به عنوان هدف قرار دادیم و به کمک باقی بردار ویژگی اقدام به تخمین آن به کمک مدل رگرسیون خطی کردیم.

تمیز سازی داده ها:

حذف مقادیر null:

نکته ای که در برخورد اولیه با دیتا ست به چشم میخورد مقادیر زیاد null در بعضی از ویژگی ها است به طور کلی:

```
telekomHybridUploadSpeed    223830
electricityBasePrice        222004
electricityKwhPrice         222004
energyEfficiencyClass       191063
lastRefurbish               188139
heatingCosts                183332
noParkSpaces                175798
petsAllowed                 114573
interiorQual                112665
thermalChar                 106506
numberOfFloors               97732
houseNumber                  71018
streetPlain                  71013
condition                    68489
yearConstructedRange         57045
yearConstructed              57045
firingTypes                  56964
facilities                   52924
floor                        51309
heatingType                  44856
totalRent                    40517
```

```
typeOfFlat                 36614
telekomUploadSpeed         33358
telekomTvOffer             32619
description                19747
serviceCharge               6909
pricetrend                  1832
```

- برای بر طرف کردن این مقادیر برای هر فیچر بسته به نوع آن و تعداد null های آن رویکرد متفاوتی در نظر گرفته شد.

فیچر هایی که بیش از ۷۰ درصد مقادیر آن ها null هستند را از دیتاست دراپ کردیم که شامل موارد زیر میباشد:

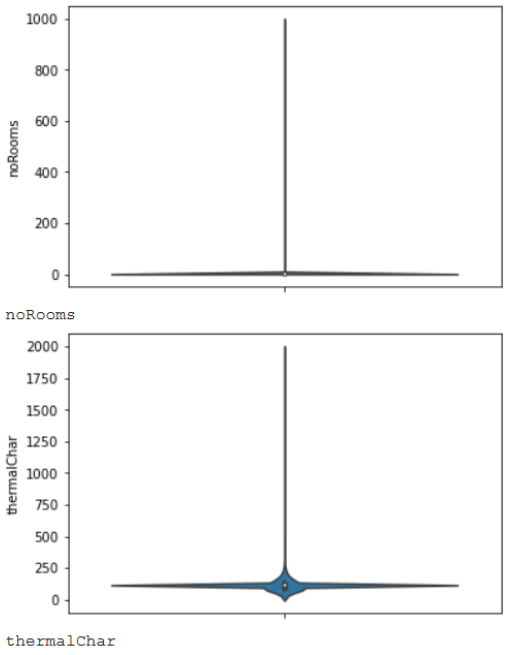
- telekomHybridUploadSpeed 223830
- electricityBasePrice 222004
- electricityKwhPrice 222004
- energyEfficiencyClass 191063

- در ابتدا برنامه داشتم که به کمک سه ویژگی ['description', 'facilities', 'streetPlain'] در جلو تر به مدل tfidf فیت بکنم که از این ویژگی ها هم در مدل استفاده کنم ولی بعد از انکود کردن فیچر های کتگوریکال و تعداد زیاد آن ها از این کار منصرف شده و این ۳ فیچر را دراپ کردم اما تا اینجای کار مقادیر نال را با اسپیس پر کردم.
- ویژگیnoParkSpaces به نظر میرسد خانه هایی که پارکینگ ندارند به ندرت مقدار صفر را ثبت می‌کنند و این فیچر برای این خانه ها null قرار میگیرد پس ما این مقدار را با صفر پر می‌کنیم
- دو ویژگی ['numberOfFloors', 'floor'] نیز مشابه ویژگی پارکینگ ایده ای مشابه داشتیم و از آن جایی که هر خانه دست کم یک طبقه است مقادیر خالی را تصور کردیم که مربوط به خانه های یک طبقه است و مقادیر null را با یک پر کردیم
- ویژگی های ['heatingCosts', 'thermalChar', 'yearConstructedRange', 'totalRent', 'telekomUploadSpeed', 'serviceCharge', 'pricetrend', ''yearConstructed'] که مقادیر عددی بودند که هیچ ایده ای در مورد نوع پر شدن آن ها به ذهن نمی‌رسید را با مقدار میانگین آن ویژگی پر کردم
- ویژگی lastRefurbish به اخرین سالی که خانه باز سازی شده است اشاره می‌کند و تصور کردیم که خانه هایی که مقدار آن ها ثبت نشده احتمالا تا به حال بازسازی نشده اند پس به کمک ویژگی yearConstructed مقادیر نال را برابر سال ساخت آن ها قرار دادیم.

- برای وژگی های کتگوریکال که مقادیر آن ها خالی بود مانند: ['petsAllowed', 'interiorQual', ''condition', 'firingTypes', 'heatingType', 'typeOfFlat', 'telekomTvOffer'] به کمک مدل درختی CATBoost و فیچر های دیگه موجود در دیتاست بغیر از فیچر هدف اقدام به پیشبینی مقادیر نال کردیم. مدل CATBoost یک نوع مدل درختی است که می‌تواند به راحتی فیچر های کتگوریکال را بدون انکود شدن به عنوان ورودی بگیرد و خروجی نیز به صورت کتگوریکال باز گرداند و از نظر قدر و سرعت از مدل های جنگل تصادفی بهتر عمل می‌کند. برای هر یک از فیچر های هدف این مدل را بر روی باقی بردار ویژگی که دارای مقدار در این ویژگی بودند آموزش دادیم و نتیجه پیشبینی را بر روی داده هایی که مقدار نال داشتند حساب کردیم و در آن ها قرار دادیم.(فیچر هدف را از این مجموعه حذف کردیم که باعث دیتا لیکیج در کار های پیش رو نشویم)

حذف مقادیر پرت:

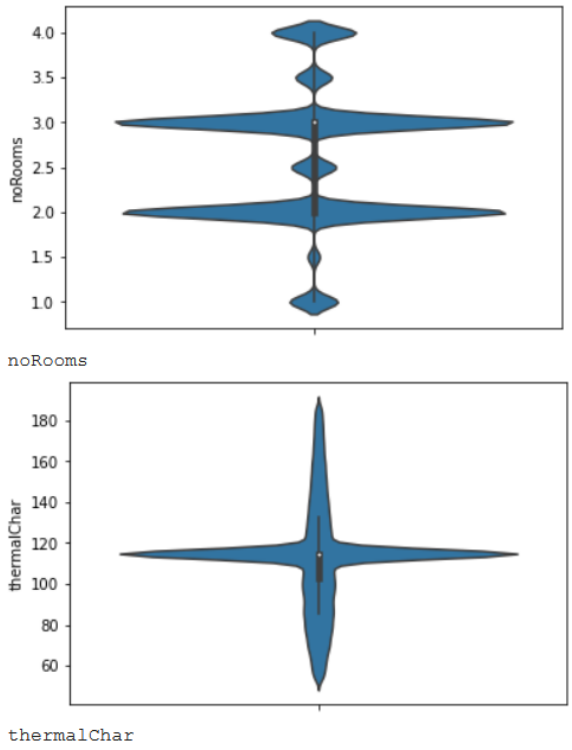
در این بخش ابتدا به مطالعه توزیع داده ها به کمک نمودار ویالون ویژگی ها پرداختیم و تمامی ویژگی هایی که به صورت زیر بودند را انتخاب کردیم :



این دم های طولانی و کشیده شده در توزیع ویژگی ها نمایان گر داده های پرت هستند و برای حذف آن ها از روش IQR با مقادری ۰.۰۵ و ۰.۹۵ استفاده کردیم. و بعد از مراحل حذف داده های پرت در هر مرحله به تعداد زیر از داده ها حذف شد:

```
serviceCharge : 268850 -> 247494
totalRent : 247494 -> 230209
yearConstructed : 230209 -> 213178
noParkSpaces : 213178 -> 206807
baseRent : 206807 -> 203218
livingSpace : 203218 -> 193393
noRooms : 193393 -> 190399
thermalChar : 190399 -> 173655
floor : 173655 -> 169881
numberOfFloors : 169881 -> 163308
heatingCosts : 163308 -> 150862
lastRefurbish : 150862 -> 142797
```

و در اخر ۱۴۲۷۹۷ داده تمیز بدون null و بدون داده پرت با توزیع های شبیه به زیر بدست آوردیم.



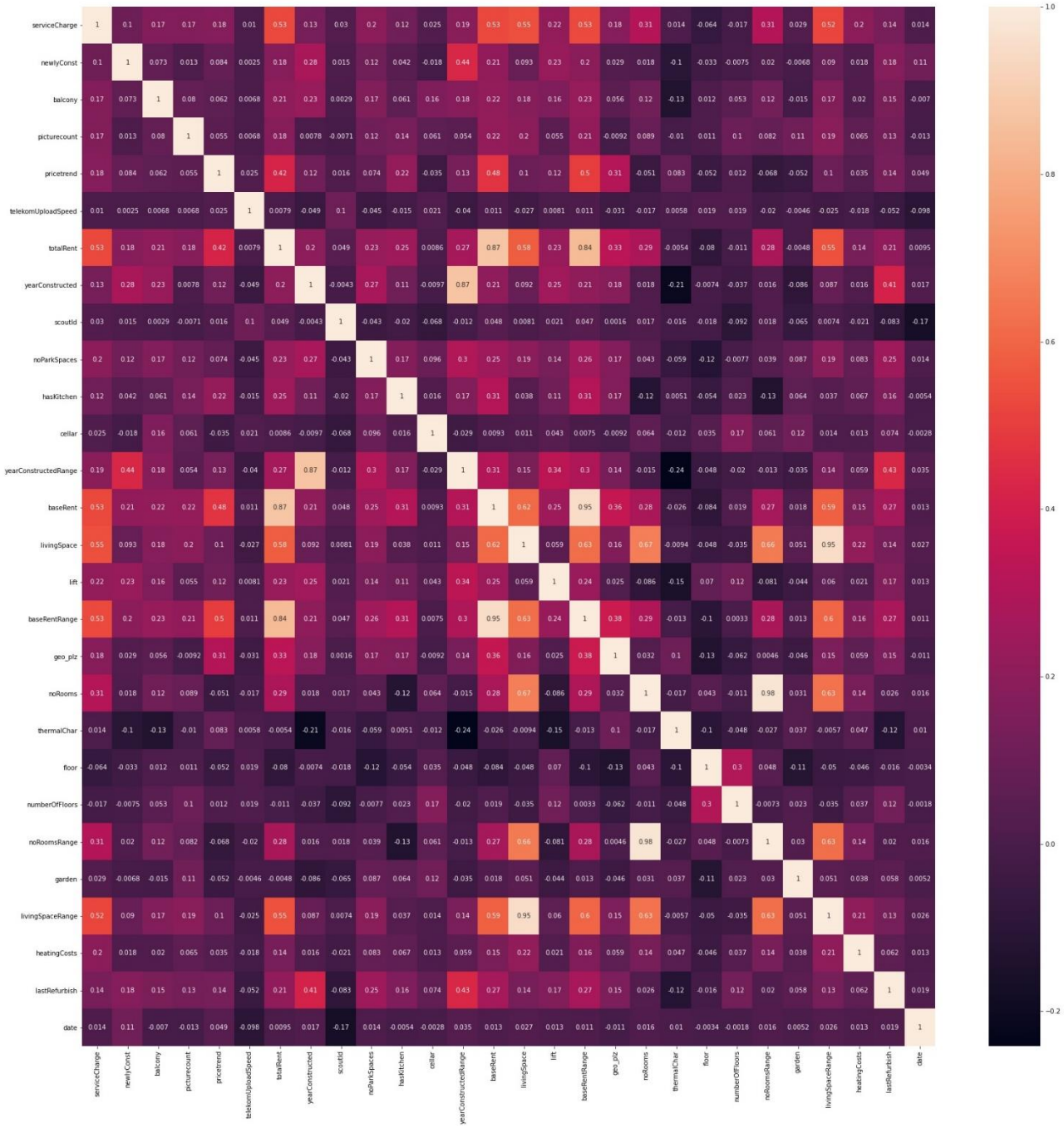
مهندسی ویژگی:

- ابتدا ویژگی ها ['newlyConst', 'balcony', 'hasKitchen', 'cellar', 'lift', 'garden', 'date''] که تنها مقادیر صحیح و غلط را شامل می‌شدند به کمک labelEncoder به صفر و یک تبدیل کردیم ویژگی date نیز تنها شامل ۴ مقدار بود و این مقادیر مقادیر ترتیبی هستند پس منطقی هست که با مقادیر ۰ و۱ و۲ و۳ به ترتیب تاریخ آن ها جایگزین شوند

- متوجه شدیم که فیچر های regio1 و regio2 دقیقا مشابه فیچر های geo_bln وgeo_krs هستند پس فقط دو ویژگی دوم را نگه داشتیم و باقی را دراپ کردیم.
- ویژگی های streetPlainو street نام خیابان ها هستند و تعداد مقادیر یکتای آن ها خیلی زیاد هست و بنظر میرسد نسبت مقدار اطلاعاتی که می‌توانند داشته باشند به هزینه محاسباتی که به سیستم اضافه می‌شوند کم است پس این دورا نیز دراپ می‌کنیم
- و در آخر ویژگی های ['heatingType', 'telekomTvOffer', 'firingTypes', 'geo_bln', 'geo_krs', 'condition', 'interiorQual', 'petsAllowed', 'typeOfFlat'] را به کمک کتابخانه pandas وان هات انکدینگ کردیم.

در پایان این بخش دیتاست اصلی ما شامل ۵۹۱ ویژگی متفاوت میشود.

سوال یک: کدام ویژگی ها بیشترین کرولیشن را با هدف دارند؟(ویژگی های خروجی از وان هات انکودینگ در نظر گرفته نشده اند)



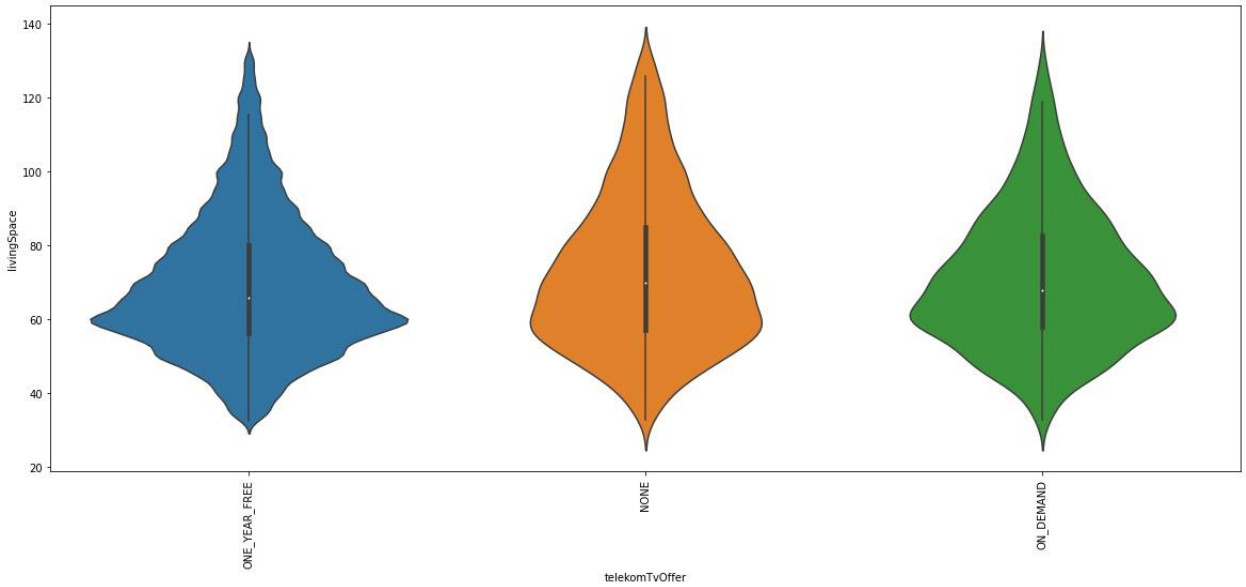
برای سادگی کار بیشترین کورولیشن را به صورت زیر هم نمایش میدهیم:

```
livingSpace             1.000000
livingSpaceRange        0.953983
noRooms                 0.668460
noRoomsRange            0.658352
baseRentRange           0.625771
baseRent                0.615193
totalRent               0.578967
serviceCharge           0.546050
heatingCosts            0.217411
picturecount            0.195358
noParkSpaces            0.193119
balcony                 0.179437
geo_plz                 0.163061
yearConstructedRange    0.149485
lastRefurbish           0.139621
pricetrend              0.102046
newlyConst              0.092899
yearConstructed         0.092301
lift                    0.059042
garden                  0.050801
hasKitchen              0.038414
date                    0.027402
cellar                  0.011291
scoutId                 0.008148
thermalChar            -0.009412
telekomUploadSpeed     -0.026983
numberOfFloors         -0.034676
floor                  -0.047704
```

همانطور که مشاهده میشود livingspaceRangeو تعداد اتاق و مقدار هزینه اجازه بیشترین کرولیسن ها را با هدف دارند و منطقی هم هست چون اگر تعداد اتاق زیاد باشد مساحت خانه افزایش میابد و خانه های بزرگ تر هزینه اجاره بیشتری هم دارند. مشاهده میکنیم که ارتباط طبیعی بین فیچر ها برقرار است پس مراحل حذف داده های پرت به درستی انجام شده است.

سوال دو: ۵ فرض آماری:

- آیا میانگین مساحت خانه در سه دسته telekomTvOfferیکسان هستند؟

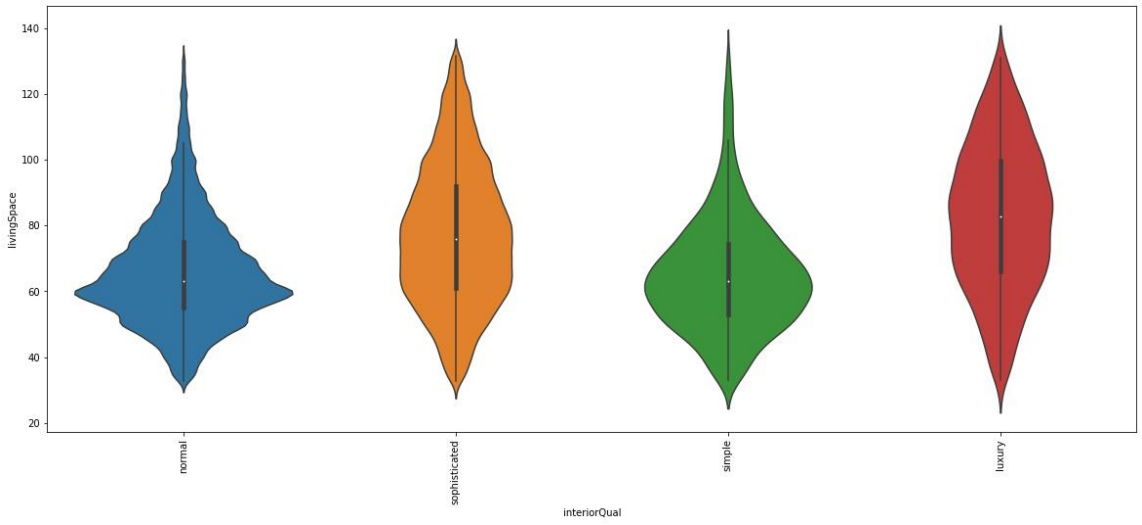


تفاوت تا حد کمی مشهود است اما با ازمون فرض Anova بررسی میکنیم:

```
(statistic=73.82257276025561, pvalue=9.032907481686397e-33)
```

مقدار pvalue کمتر از ۰.۰۵ است پس فرض صفر رد شده و میانگین ها متفاوت هستند

- آیا میانگین مساحت خانه در چهار دسته interiorQual یکسان هستند؟

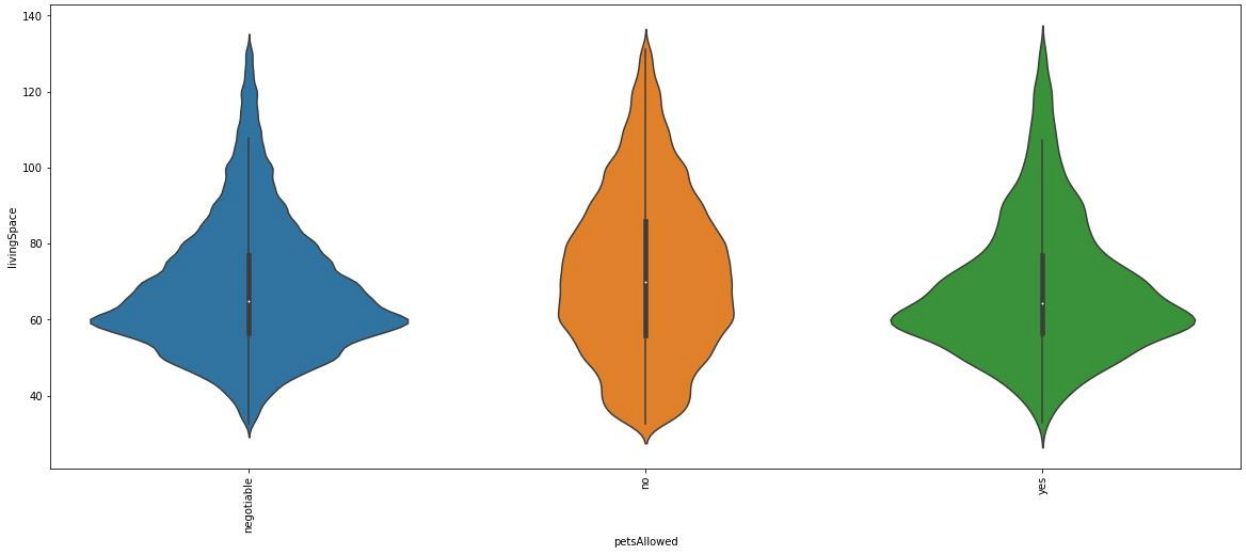


با ازمون فرض Anova بررسی میکنیم:

```
(statistic=4225.957452550397, pvalue=0.0)
```

مقدار pvalue کمتر از ۰.۰۵ است پس فرض صفر رد شده و میانگین ها متفاوت هستند

- آیا میانگین مساحت خانه در سه دسته petsAllowed یکسان هستند؟

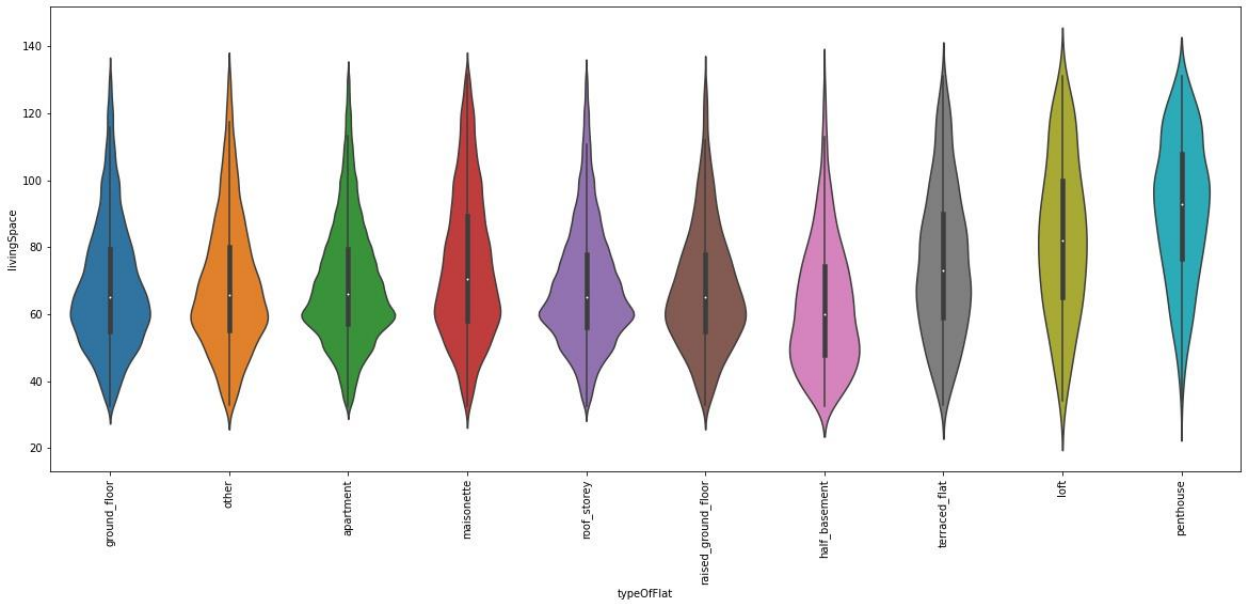


با ازمون فرض Anova بررسی میکنیم:

```
(statistic=723.4140244826011, pvalue=2.5483730514e-313)
```

مقدار pvalue کمتر از ۰.۰۵ است پس فرض صفر رد شده و میانگین ها متفاوت هستند

- آیا میانگین مساحت خانه در ده دسته typeOfFlat یکسان هستند؟



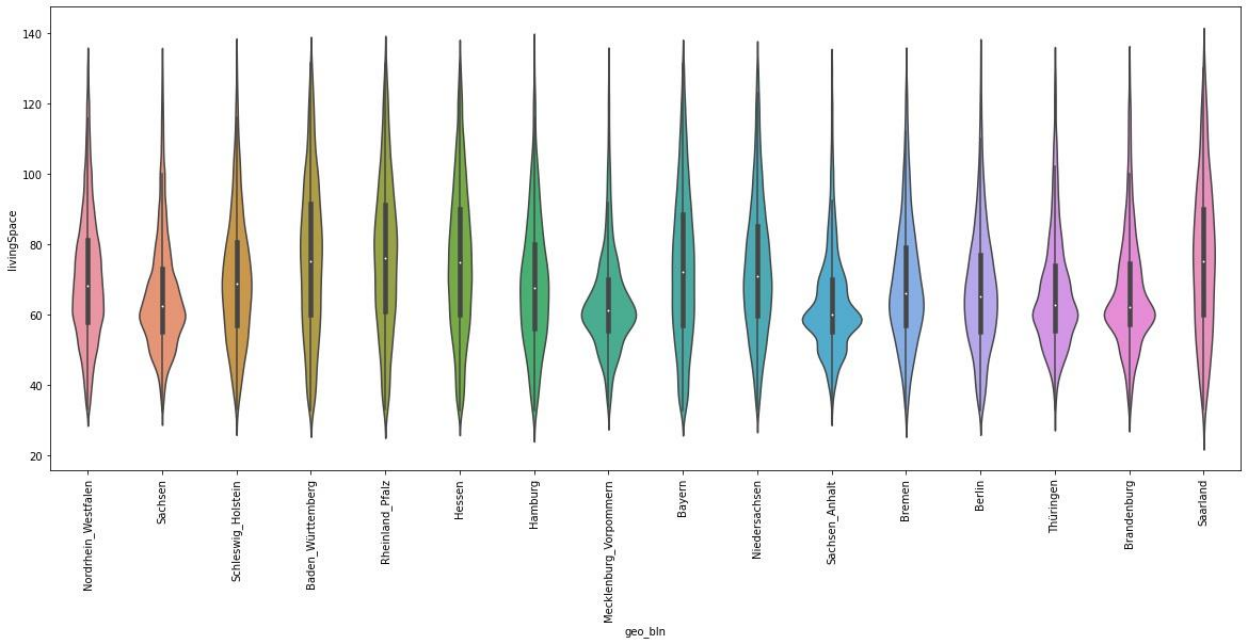
با ازمون فرض Anova بررسی میکنیم:

```
(statistic=279.2374096482586, pvalue=0.0)
```

مقدار pvalue کمتر از ۰.۰۵ است پس فرض صفر رد شده و میانگین ها متفاوت هستند

- آیا میانگین مساحت خانه در شانزده دسته regio1/geo_bln یکسان هستند؟

سوال اصلی این هست که آیا محله ها و شهر ها از نظر میانگین مساحت خانه های یکسان هستند یا در برخی از مناطق میانگین کل خانه ها کم و در برخی دیگر زیاد است



با ازمون فرض Anova بررسی میکنیم:

```
(statistic=462.18284356789746, pvalue=0.0)
```

مقدار pvalue کمتر از ۰.۰۵ است پس فرض صفر رد شده و میانگین ها متفاوت هستند

همانطور که در نمودار هم مشاهده می‌شود بعضی از مناطق از نظر میانگین مساحت خانه نزدیک به هم هستند

حال روی همین مشاهده یک سوال مطرح میکنیم

- آیا میانگین مساحت خانه های Brandenburgو Thüringenو Berlin یکسان هستند؟

به کمک آزمون anova بررسی میکنیم:

```
(statistic=1.4698083246647065, pvalue=0.23001007721429811)
```

مقدار value بزرگتر از ۰.۰۵ است پس بله فرض درست بوده و میانگین مساحت خانه های این ۳ منطقه یکسان است پس میتوان گفت بعضی خوشه ها در نوع خانه های هر منطقه وجود دارد و بعضی مناطق مانند سه منطقه بالا از نظر بافت شهری شبیه به هم هستند.

مدل سازی:

برای مدل سازی ابتدا دیتا ست کامل را به دو مجموعه آموزش و تست با نسبت ۷۰ به ۳۰ تقسیم کردیم سپس سه مجموعه داده تولید کردیم اولی شامل همه فیچر ها دومی شامل ۱۰ فیچر اصلی استخراج شده از pca (عدد ده بعد از تحلیل cumulative explained variance انتخاب شده) و سومی هفت فیچر که بیش از ۰.۵ با هدف کورلیشن دارند شامل ('livingSpaceRange', 'noRooms', 'noRoomsRange', 'baseRentRange', 'baseRent', 'totalRent', 'serviceCharge') میباشد.

سوال سه: پیاده سازی از پایه

در این بخش دو مدل پیاده سازی کردیم یکی از آنها که به کمک Ordinary least squares اقدام به حل مسئله رگرسیون خطی میکند و در دل خود تابع MSE را بهینه میکند.

```
class My_LinearRegressin():
    def __init__(self):
        self.coefficients = None
        self.intercept = None

    def fit(self, x, y):
        x = np.c_[np.ones(len(x)),x]
        y = y.values

        xT = x.transpose()
        inversed = np.linalg.inv( xT.dot(x) )
        betas = inversed.dot( xT ).dot(y)

        self.intercept = betas[0]
        self.coefficients = betas[1:]

    def predict(self, x):
        return np.multiply(x, self.coefficients).sum(axis = 1) +
self.intercept
```

مدل دیگری که پیاده سازی کردیم به کمک gradient descent اقدام به حل کردن مسئله رگرسیون میزند

در این مدل تابع هدف را تابع epsilon-sensitive در نظر گرفتیم چون اگر در این مدل مقدار اپسیلون برابر صفر قرار بگیرد مشابه این هست که تابع MAE را بهینه میکند پس این انعطاف را داریم که دو تابع هزینه را با یک مدل حل کنیم.

```
class My_LinearRegressin_epsilon_sensitive():
    def __init__(self, epsilon=0):
        a = None
        b = None
        self.epsilon = epsilon

    def predict(self, X):
        return X.dot(self.a)+self.b

    def fit(self, X, Y, alpha = 0.01, iterations = 500):

        self.a = np.zeros(X.shape[1])
        self.b = 1

        n = len(X)
        print(self.epsilon)
        for iteration in range(iterations):
                h = self.predict(X)

                if (Y - h).mean() > self.epsilon:
                    a_gradient = -(1/n)#*X.sum(axis = 0)
                    b_gradient = -1/n
                elif (h - Y).mean() > self.epsilon:
                    a_gradient = (1/n)#*X.sum(axis = 0)
                    b_gradient = 1/n
                else:
                    a_gradient = 0
                    b_gradient = 0
                self.a = self.a - alpha * a_gradient
                self.b = self.b - alpha * b_gradient
```

سوال پنج: رگرسیون به کمک پکیج ها

در این سوال از سه مدل به کمک دو کلاس از پکیج sklearn ساختیم که یکی بر پایه MSE و یکی بر پایه MAE و یکی هم epsilon_sensitive است

`LinearRegression()`	MSE
`make_pipeline(StandardScaler(),LinearRegression()`	MSE+Scaler

`make_pipeline(StandardScaler(),`
`SGDRegressor(max_iter=1000, tol=1e-3,`
`loss = 'epsilon_insensitive',`
`epsilon=0))`
	MAE+Scaler
	+SGD

`make_pipeline(StandardScaler(),`
`SGDRegressor(max_iter=1000, tol=1e-3,`
`loss = 'epsilon_insensitive',`
`epsilon=0.5))`
	Epsilon+Scaler
	+SGD

سوال چهار و شش: نتایج فیتینگ مدل های دست ساز و کتابخانه ای را در زیر بر اساس ملاک های خواسته شده در صورت سوال اماده کردیم:

Train						
مدل	دیتاست	MSE	MAE	Accuracy (%) with 10% margin	R2	RSE
My_MSE	All-Features	23.88	4.03	82	0.93	4.88
My_MSE	Corr_Features	26.44	4.28	79	0.92	5.14
My_MSE	PCA_Features	187.32	10.36	43	0.46	13.68
My_MAE	All-Features	424.74	15.81	27	-0.21	20.60
My_Epsolon	All-Features	424.74	15.81	27	-0.21	20.60
MSE	All-Features	23.87	4.03	82	0.93	4.88
MSE+Scaler	All-Features	23.88	4.03	82	0.93	4.88
MSE+Scaler	Corr_Features	26.44	4.28	79	0.92	5.14
MSE+Scaler	PCA_Features	187.32	10.36	43	0.46	13.68
MAE+Scaler+SGD	All-Features	24.82	3.99	82	0.92	4.98
Epsilon+Scaler+SGD	All-Features	24.79	3.99	82	0.92	4.97

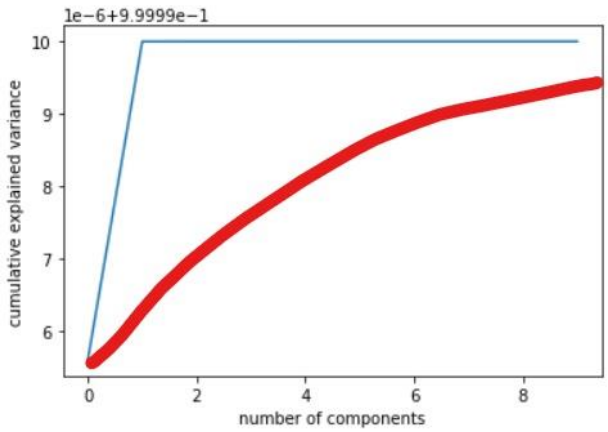
Test						
مدل	دیتاست	MSE	MAE	Accuracy (%) with 10% margin	R2	RSE
My_MSE	All-Features	24.43	4.07	82	0.93	4.94
My_MSE	Corr_Features	26.57	4.29	79	0.92	5.15
My_MSE	PCA_Features	188.29	10.36	43	0.46	13.72
My_MAE	All-Features	434.95	15.94	27	-0.22	20.85
My_Epsolon	All-Features	434.95	15.94	27	-0.22	20.85
MSE	All-Features	7431.64	5.21	82	-20	86.20
MSE+Scaler	All-Features	9.34	344613289	81	-2.64	30566866314
MSE+Scaler	Corr_Features	26.57	4.29	79	0.92	5.15
MSE+Scaler	PCA_Features	188.29	10.36	43	0.46	13.72
MAE+Scaler+SGD	All-Features	25.4	4.05	82	0.92	5.03
Epsilon+Scaler+SGD	All-Features	25.29	4.05	82	0.92	5.02

مدل	دیتاست	Time
My_MSE	All-Features	5.85s
My_MSE	Corr_Features	403 ms
My_MSE	PCA_Features	330 ms
My_MAE	All-Features	1min 48s
My_Epsolon	All-Features	1min 54s
MSE	All-Features	3.36 s
MSE+Scaler	All-Features	5.51 s
MSE+Scaler	Corr_Features	200 ms
MSE+Scaler	PCA_Features	177 ms
MAE+Scaler+SGD	All-Features	8.05 s
Epsilon+Scaler+SGD	All-Features	7.15 s

نتیجه گیری و تحلیل نتایج:

اختلاف زیادی بین نتایج حاصل بر روی مجموعه داده تست و آموزش وجود ندارد پس می‌توان نتیجه گرفت مدل ها به قدر مناسبی عمومی هستند.

نتایج بر روی فیچر های pca ضعیف تر عمل کرده اند و این اتفاق به این دلیل است که یکی از فیچر ها بخش زیادی از واریانس داده ها را شامل میشود و نمودار آن به شکل زیر است(ابی رنگ):



نتایج pca اکثرا وقتی خوب عمل میکنند که نمودار آن به صورت قرمز رنگ باشد که در این مسئله اینگونه نیست.

مدل های پیاده سازی شده از نظر دقت اختلاف زیادی با مدل های کتابخانه ای در بخش نتایج ندارند ولی اما از نظر زمانی نسبتا ضعیف عمل کرده اند و این به این دلیل است که کتابخانه sklearn بعضی محاسبات را در سطح c انجام میدهد و سرعت قابل قبول تری دارد.

مدل هایی که r2 منفی کسب کرده اند به این معنی است که مدل از مدل ساده ای که تنها از میانگین به عنوان خروجی استفاده کند هم بدتر عمل کرده است. دقت پایین دو مدل MAE و Epsilon به دلیل مشکلات پیاده سازی گرادیان دیسنت معمولی می‌باشد که در این نوع از مسائل جوابگو نیست و باید از روش های بهینه سازی مانند adam و ... استفاده کرد تا اثر داده های زیاد و مقادیر پرت که برای تابع های هزینه MAE و Epsilon اثر منفی دارند خنثی شود.

صحت پیاده سازی مدل ها بر روی مثال ساده دو بعدی امتحان شده و درست میباشند.

نمونه آزمایش صحت مدل های پیاده سازی شده.