به نام خدا

گزارش تمرین سری اول – قسمت دوم درس داده کاوی

نام و نام خانوادگی: پویا شاعری

شماره دانشجویی: 400422105

لینک Colab:

https://colab.research.google.com/drive/1VC4NuUqWlhCnXcjRCLdPPFqMLHgoz4AE?usp=sharing

چکیده

در این گزارش قصد داریم مرحله به مرحله کارهای انجام شده را توضیح دهیم و تحلیل کنیم. گرچه در داخل نوت‌بوک، به صورت تکست بالای هر سلول توضیح داده شده است ولی ضروری است توضیحات و تحلیل های لازم و مقایسه ها را در اینجا داشته باشیم.

مقدمه

در ابتدا با توجه به آموزش داده شده در صورت تمرین، دیتاست، توسط سلول هایی، از سایت کگل مستقیم خوانده و در df_original ذخیره شد. آن را نمایش دادم و نگاهی اجمالی به داده کرده و از پایین سمت چپ دیتاست، shape آن را نگاه کردم. در سلول بعدی بااستفاده از دستور pd.options.display.max_columns تعداد ستون های نمایشی را بیشتر کردیم تا کل داده ها را ببینیم. البته داخل خود کگل که دیتاست قرار دارد می‌توانیم ستون به ستون داده ها رو ببینیم و هر ستون توضیح هم دارد به همراه نمودار. می‌بینیم که با داده‌های نسبتا بدی سر و کار داریم. در توضیحات مربوط به این دیتاست، میفهمیم که 41٪ آن missing است.

در سلول بعد از روی آن کپی گرفتم و از آنجا به بعد روی کپی به نام df کار کردم، این کار هم با copy(). و هم با assign کردن (تا یکبار) امکان‌پذیر است. این کار به این دلیل است که اگر جایی اشتباه شد و df از دست رفت، از سلول کپی گرفتن دوباره کار را از سر بگیرم و به بالای بالا مراجعه نکنم.

بخش اول: پیش‌پردازش داده ها

در سلول بعدی بااستفاده از تابع info(). روی df، اطلاعات کلی از اسم ستون ها و تعداد غیر null ها و Dtype داده ها (در هر ستون) و وضعیت حافظه را مشاهده کردم. با دستور df.isna().sum()/len(df) می بینم که چند درصد هر فیچر میسینگ است. در ابتدا با روش ZScore داده های پرت را بیرون ریختم چون تقریبا داده ها کلان هست و پر از میسینگ و داده‌های پرت و با اینکار حداقل با یک سری داده های بهتری از ابتدای کار سر و کار داشته باشم (برای مثال ستون هدف totalRent دارای برخی داده های میلیون دلاری بود که با توضیحات شما این داده های پرت و بعضی داده های null باید به شخص کارفرما گزارش می شد. حال در ادامه ستون هایی که صفر بودن آنها مهم است را بررسی کردم:

```
[ ]  1 len(df[df['livingSpace'] == 0])
```

```
67
```

```
[ ]  1 len(df[df['baseRent'] == 0])
```

```
79
```

```
[ ]  1 len(df[df['yearConstructed'] == 0])
```

```
0
```

```
[ ]  1 len(df[df['geo_plz'] == 0])
```

```
0
```

مثلا فضا، سال ساخت و ...

و مهم تر از همه totalRent:

```
[ ]   1 len(df[df['totalRent'] == 0])
```

```
223
```

صفر های بقیه رو از اونجا که کم هستند، دراپ کردم ولی 0 های totalRent رو نگه داشتم و نال در نظر گرفتم و قرار هست که با منطق مناسبی آن ها را پر کنم. منطق مناسب این دو راه است:

1- رابطه ای بین baseRent و serviceCharge و قبض های برق و گرما:

totalRent 40517 ta Missing dare yani 15% !!! ino Chejori Por konim ?

$$total\ rent = base\ rent + N_1 \times ghabze\ electricityo\ heating\ ina + N_2 \times service\ charge$$

inaro migiram Noise dar (N_1 , N_2) ba Ham Jam mikonam

2- امتیازی: با استفاده از مدل لرن کنم و چقدر نتایج خوبی به همراه داشت. ولی ایرادی که به آن وارد است، این است که داده ها نسبت به مدل bias می شوند به اصطلاح و این خوب نیست. شاید اگر ستون دیگری که در آینده قرار نبود target قرار گیرد را با چند ستون دیگر از این راه پر کنیم بهتر می بود.

در ادامه به توضیح هریک این روش ها می پردازیم.

به ادامه کار باز گردیم.

به هندل کردن سطر های duplicate با استفاده از یک سری ستون های reliable پرداخته ایم. ستون هایی را reliable دانسته ایم که اولا نال نداشته باشند، دوما مشخصات مربوط به هر خانه را دقیقا بیان کنند و سوما تعداد این ستون ها را به حدی زیاد گرفتیم که ماسک خوبی برای مشابهت ها ارائه کنند.

Duplicated Handling

```
[ ]  1 df = df.drop(df[df['livingSpace'] == 0].index)
     2 df = df.drop(df[df['baseRent'] == 0].index)
```

```
1 duplicatedMask = df.duplicated(subset=['regio1', 'regio2', 'regio3','geo_plz', 'geo_krs',
2                                        'livingSpace', 'baseRent', 'newlyConst', 'cellar', 'hasKitchen',
3                                        'balcony', 'garden', 'lift', 'noRooms'], keep=False)
4 len(duplicatedMask)
```

```
247640
```

```
[ ]  1 df = df[duplicatedMask == False]
```

```
[ ]  1 df.shape
```

```
(222150, 49)
```

تعداد ستون ها نسبتا زیاد است و باید کم شود.

در ادامه می‌خواهیم ستون هایی که بیش از 50٪ آن ها نال است را دراپ کنیم.

Mikham Sotunayi k Bishtaraz 50% eshon Nulle Drop Konam

```
[ ]  1 df.columns[df.isna().sum()/len(df) > 0.50]
```

```
Index(['telekomHybridUploadSpeed', 'noParkSpaces', 'heatingCosts',
       'energyEfficiencyClass', 'lastRefurbish', 'electricityBasePrice',
       'electricityKwhPrice'],
      dtype='object')
```

vali fln ghabzaye ***Electricity*** va ***Heating*** ro mikhameshon , baraye hamin negaheshon midaram hatta age ba mean Por konim ya harchi baz baraye mohasebate Dakhele Model **Reliable Nis**.

pas ghablaz Model zadan too Taske Akhar Drop Mishan.

پس فعلا قبض برق و گاز را نگه می دارم و بعدا قبل از مدلسازی دراپ میکنم و فعلا فقط ستون‌هایی غیر از این قبض ها را دراپ کردم که بالای 50٪ نال داشتند. بررسی می کنیم که چه ستون های دیگری قابل حذف هستند.

Sotun Hayi k mikham Drop She (Numerical):

'houseNumber' : Chon null dare , Chejori masalan Poresh konam ???

Range ha: **'yearConstructedRange', 'baseRentRange', 'noRoomsRange', 'livingSpaceRange'**

Sotun hayi k mikham Drop She (Categorical):

'Onayi k bishtar az 16 ta hastan'

Discription ha k Almani Neveshte: **'description', 'facilities'**

'geo_bln' Chon tekrare Regio 1

boodan naboodane 'date' ba 4 ta done che Tasiri dare

مثلا پلاک خانه دارای جنس استرینگ (مثلا 40A) چجوری پر بشه. Range ها رو من خودم به عنوان دیتا ساینتیست میتونم استخراج کنم (چرا باید یکی بیاد بگه در چه رنج فضایی هست این فضای خونه وقتی خود همین ستون رو دارم). برای حذف ستون های categorical، دیتاست آنها استخراج کردم و nunique زدم.

```
1 df_cat = df.select_dtypes(exclude=["number"])
2 df_cat.nunique()
```

```
regio1               16
heatingType          13
telekomTvOffer        3
newlyConst            2
balcony               2
firingTypes         127
hasKitchen            2
geo_bln              16
cellar                2
geo_krs             419
condition            10
interiorQual          4
petsAllowed           3
street            48016
streetPlain       49676
lift                  2
typeOfFlat           10
garden                2
regio2              419
regio3             8331
description      183472
facilities       161127
date                  4
```

و اونایی که بیشتر از 16 نوع داشتند رو هم پاک کردم. دقت کنید که باز هم نیاز به حذف ستون هایی مانند date که کلا 4 نوع ازش داریم و به مدلسازیمون هم کمک نمیکنه و کدپستی محله geo_plz داریم ولی شاید در تحلیل آماری و رسم نمودار و ... در تسک 2 نیازمند به وجود آنها باشیم و مانند قسمت قبل تمرین آن ها را قبل از مدل حذف میکنیم.

توضیح حالت 1: پر کردن null های totalRent با استفاده از رابطه گفته شده:

totalRentهایی که نال هستند را صفر میکنیم. که به همراه صفر ها پر کنیم.

Berim Soraghe Total Rent k Targetemone

```
[ ]  1 df.totalRent.isna().sum()
```

```
33464
```

```
[ ]  1 df.totalRent.isna().sum()/len(df)
```

```
0.15063695701102858
```

```
[ ]  1 df.totalRent.fillna(0, inplace = True)
```

```
[ ]  1 (df.totalRent == 0).sum()
```

```
33647
```

قبض گاز و برق رو با میانگین پر کردیم (میدونم نال زیاد داشت ولی مجبورا این کار را کردیم) و با در نظر گرفتن یک نویز به عنوان ضریب برای قبض و نویز دیگر ضریب ستون serviceCharge که نال 2٪ی داشت و خیلی خوبه و خود baseRent که نال نداره عالیه و جمع این سه مورد totalRent هایی که صفر بودند را پر کردیم (چون در ابتدای کار نال ها رو هم صفر کردیم)، پس درواقع نال ها رو با یه منطق درستی پر کرده ایم. این کار را این گونه انجام داده ایم که df_tr0 به عنوان دیتافریمی که totalRent آنها صفر است در نظر گرفتم و df_tr0 را با رابطه پر کردم و سپس df_trna0 را به عنوان دیتافریمی که totalRent آنها صفر نیست در نظر گرفتم و این دو دیتافریم را به هم concat کردم.

می بینیم که جایی totalRent نال/صفر نداریم:

```
[ ]   1 (df.totalRent == 0).sum()

0
```

سپس در سلول بعدی info گرفتم و null های باقی مانده عددی را با میانگین و categoricalرا با مد آن ها پر کرده ام. و می‌بینیم که دیگر کلا نال نداریم.

```
[ ]   1 df.isna().sum().sum()

0
```

مجموعه سلول های بعدی: حالت 2: امتیازی با استفاده از لرن کردن totalRent رو تخمین بزنیم.

▸ ----------LEARN AND IMPUTE TOTAL RENT---------

```
[ ]  ↳ 22 cells hidden
```

می توانیم از حالت 1 استفاده نکنیم و فقط تا قبل از حالت 1 پیش برویم و سپس به سراغ این قسمت برای پر کردن داده های نال totalRent بیاییم و سپس نال های باقی مانده را میانگین و مد کنیم. پس در هنگام ران گرفتن کد به این موضوع توجه بفرمایید.

در این قسمت ابتدا تمامی ستون های عددی رو در نظر گرفتم و میانگین زدم روی نال ها و سپس بعد از نرمال سازی و آت لایر دیتکشن، رگرسیون خطی رو با y=totalRent انجام دادم. به طوری که داده های تستم رو اونایی گرفتم که صفر بودن (نال ها + صفر ها). میتونستم یه مقداری از train رو اختصاص بدم به validation ولی بدون validation این کار رو انجام دادم.

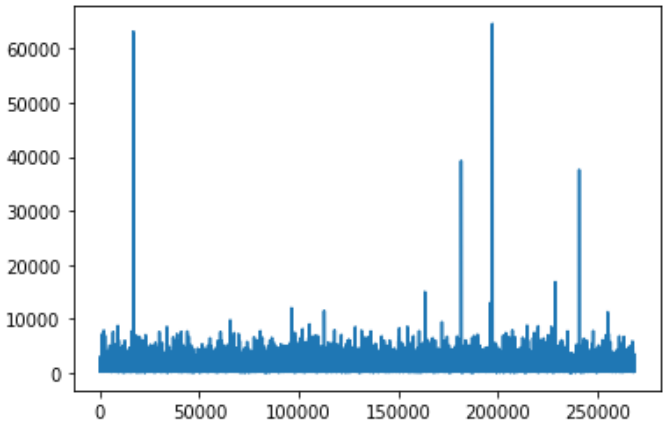
بخش دوم: تحلیل آماری و مصورسازی داده ها

با دستور describe() کل ستون های عددی بررسی شد : مینیمم ماکزیمم میانگین و ... . اگر ستون های categorical داشتیم که داریم باید حواسمان باشد که روی این ستون ها مجزا describe بزنیم که مد و فراوانی و ... را ببینیم که با حلقه روی ستون های categorical زده‌ایم.

```
1 cols = []
2 for cols in df.columns:
3   if df[cols].dtype == 'bool' or df[cols].dtype == 'object':
4     print(df[cols].describe())
5     print('-----------------------')
```

```
top         True
freq      136638
Name: balcony, dtype: object
-----------------------
count     222150
unique         2
top        False
freq      143892
Name: hasKitchen, dtype: object
-----------------------
count     222150
unique         2
top         True
freq      142053
Name: cellar, dtype: object
-----------------------
count        222150
unique           10
top       well_kept
freq         113046
Name: condition, dtype: object
```

نمودار توزیع totalRent را با دستور پلات میبینیم.



این نشان دهنده آن است که هنوز outlier داریم. با جستجویی که انجام دادم در سایت stackoverflow آورد که میتوان دوبار Z-Score و Outlier Detection انجام داد.

2

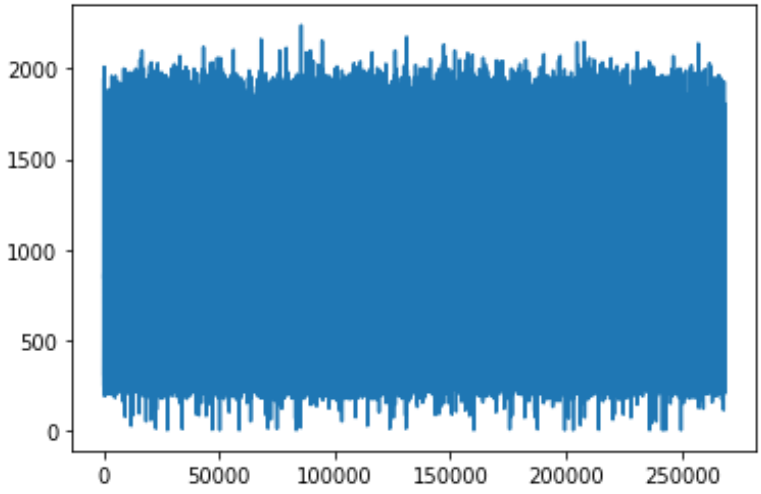
The z-score tells you how many standard deviations away from the mean a certain point is. Using |z-score| > 3 is a very common way to identify outliers. What you are missing, is that when you remove/replace outliers, the standard deviation of your new distribution is different than it used to be, thus the z-scores of all remaining points are slightly different. In many cases, the change is negligible; however, there are some cases where the change in z-score is more pronounced.

Depending on your application, you may wish to run the z-score filter a couple times until you get a stable distribution. Also, depending on your application, you may consider dropping outlier data instead of replacing them with the median. Hopefully you know why you chose to replace and the caveats associated with that choice.

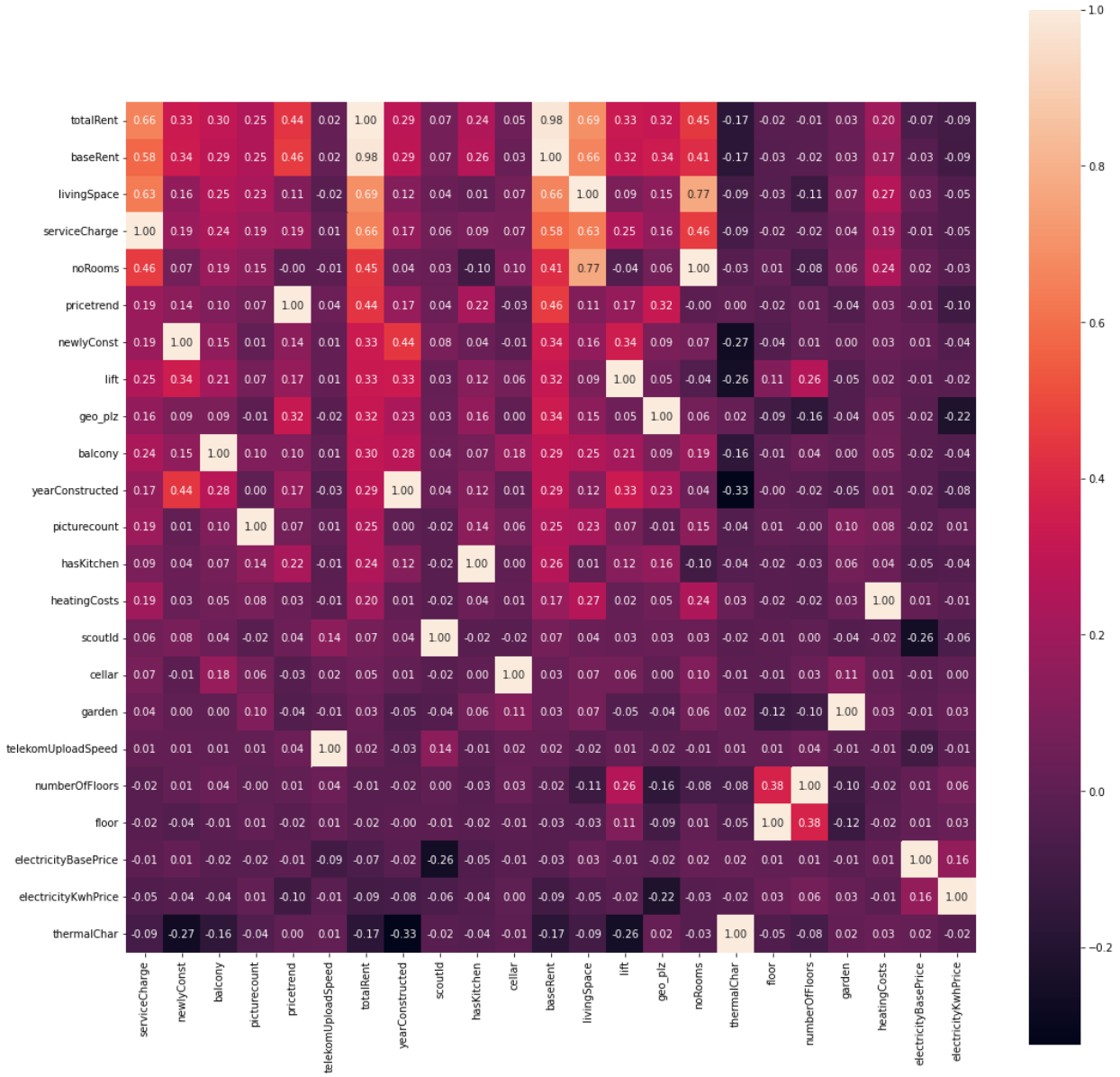
Share Follow

answered Jun 19, 2021 at 15:03

و با انجام دوباره Z-Score داریم:



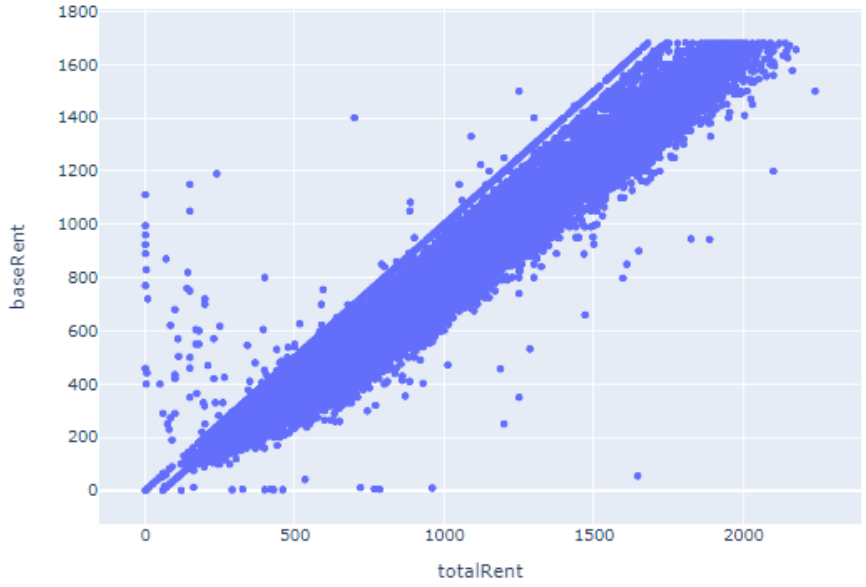
نمودار (ماتریس) کوریلیشن را نیز با دستور heatmap از sns رسم کرده ایم.

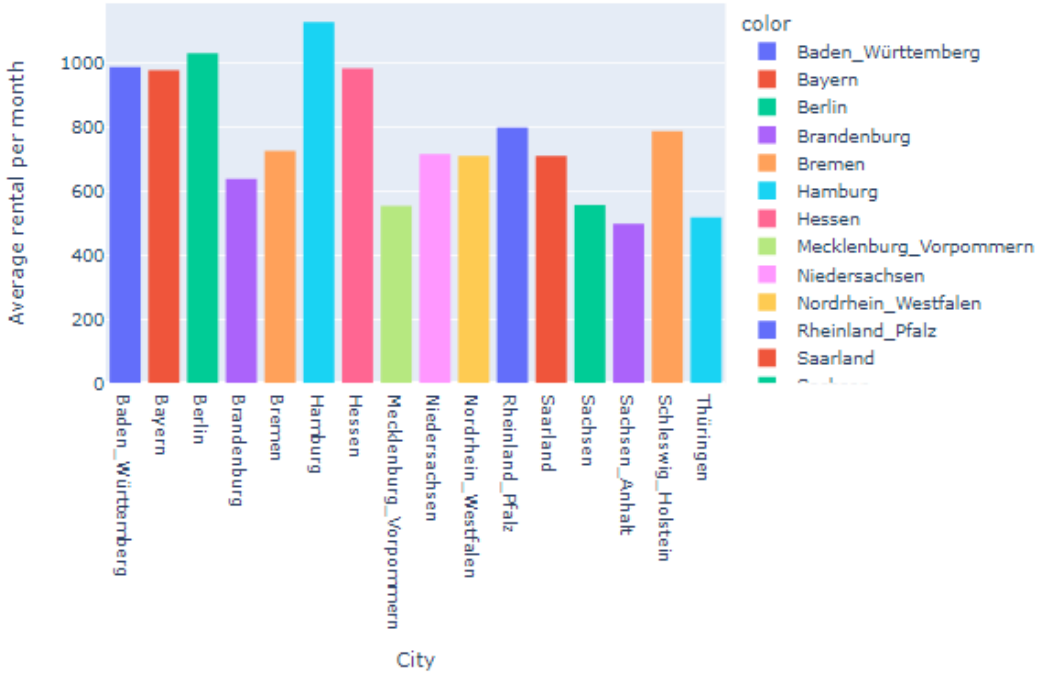


از این نمودار می توان دریافت که totalRent با baseRemt همبستگی زیاد و تعداد اتاق ها با فضای خانه نیز بعد از آن دارای کوریلیشن بالا هستند.

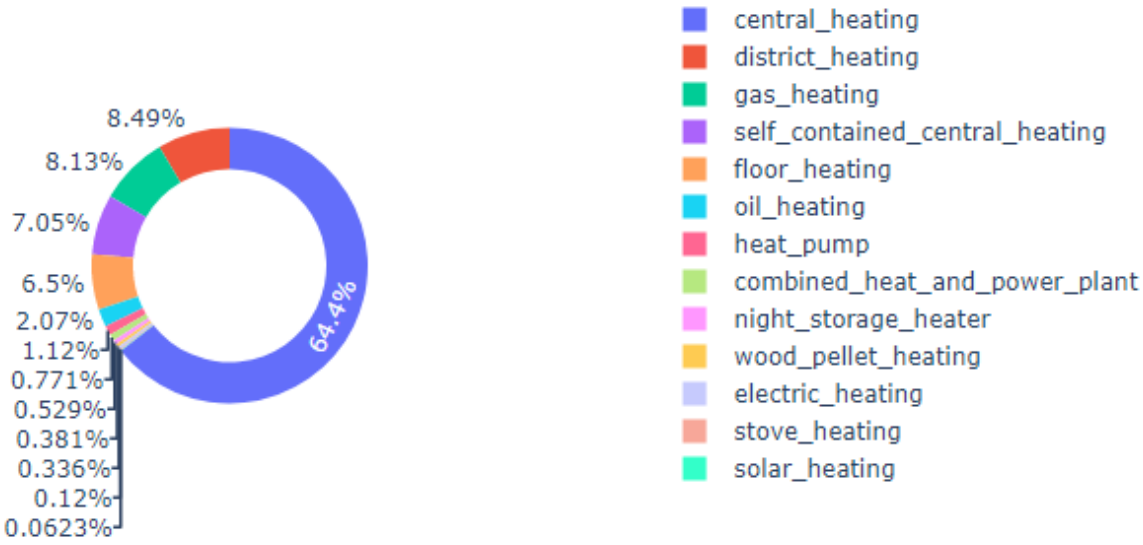
در ادامه نمودار totalRent بر حسب baseRent را داریم و از plotly با متد histogram و با استفاده از groupby نمودار میله ای Average rental per month بر حسب City را داریم.

```
1 import plotly.express as px
2 fig = px.scatter(df, x='totalRent', y='baseRent')
3 fig.show()
```





نمودار pieChart شهر ها برحسب نوع سیستم گرمایشی heatingType



این کار نیز با یک ولیوکانتز روی heatingType و متد countPie و iplot انجام گرفته است.

و نمودار های دیگر که در کد مشخص است.

بخش سوم: مدلسازی

ابتدا در این بخش داده های categorical را عددی می کنیم با وانهات و گت دامیز (در گزارش قسمت قبل توضیح دادم) و قرار بود ستون های قبض برق و geo_plz وscout_ID و ... حذف شوند، چون در مدل بی ارزش هستند. Shape میگیریم میبینیم 75 تا فیچر داریم پس حتما PCA می خواهیم. ابتدا y و x را جدا می کنیم و سپس چون قراره PCA بزنیم روی یک کپی به اسم df2 کار کردم. PCA با components = 20 زدم و با استفاده از یک مدل رگرسیون خطی در sklearn و نرمال کردن و اسپیلیت کردن test و train مدلسازی را انجام داده ام و در نهایت y test را نیز predict کردم.

بخش چهارم: multiprocessing و runtime

برای انجام کار، df را ریست کرده و می خواهیم Z-Score را در multiprocess انجام دهیم.

Time را import و از multiprocessing ، Pool و cpu_count را نیز import میکنیم.

```
[113]  1 s_time = time.time()
       2 for cols in df.columns:
       3     if df[cols].dtype == 'int64' or df[cols].dtype == 'float64':
       4         upper_range = df[cols].mean() + 3 * df[cols].std()
       5         lower_range = df[cols].mean() - 3 * df[cols].std()
       6
       7         indexs = df[(df[cols] > upper_range) | (df[cols] < lower_range)].index
       8         df = df.drop(indexs)
       9 print(f'runtime : {time.time() - s_time}')
```

```
runtime : 2.6160495281219482
```

```
1 df.shape
```

```
(247777, 49)
```

```
[115]  1 df = df_original
       2 df.shape
```

```
(268850, 49)
```

```
[116]  1 def myzscore(col, df = df):
       2   if df[col].dtype == 'int64' or df[col].dtype == 'float64':
       3     upper_range = df[col].mean() + 3 * df[col].std()
       4     lower_range = df[col].mean() - 3 * df[col].std()
       5
       6     indexs = df[(df[col] > upper_range) | (df[col] < lower_range)].index
       7     df = df.drop(indexs)
       8     return df
```

```
1 s_time = time.time()
2 with Pool(cpu_count()) as p:
3
4   df_list = p.imap(myzscore, df.columns)
5
6 print(f'runtime : {time.time() - s_time}')
```

```
runtime : 0.284625768661499
```

می بینیم که چقدر runtime کاهش یافت.

بخش پنجم: multiprocessing و runtime

df را ریست کرده و می خواهیم پر کردن nullها و outlier را در Dask انجام دهیم.

```
1 !python -m pip install "dask[complete]"
```

```
Requirement already satisfied: dask[complete] in /us
Requirement already satisfied: toolz>=0.7.3 in /usr/
Requirement already satisfied: partd>=0.3.10 in /usr
```

از کتابخانه مربوطه، ایمپورت های لازم را انجام می دهیم:

```
[12]  1 import dask
      2 import dask.dataframe as dd
```

```
[41]  1 df = df_original
```

```
      1 df.shape
```

```
(268850, 49)
```

```
[43]  1 ddf = dd.from_pandas(df, npartitions = 4)
```

```
[44]  1 ddf.partitions[0]
```

Dask DataFrame Structure:

	regio1	serviceCharge	heatingType	telekomTvOffer	telekomHyb
npartitions=1					
0	object	float64	object	object	

داده را ریست میکنیم و از df ، ddf را که دسک دیتافریم است ، میسازیم و آنرا چهار پارتیشن میکنیم.

```
[47]  1 s_time = time.time()
      2 df.fillna(df._get_numeric_data().mean(),inplace = True)
      3 for column in df.columns:
      4   if df[column].dtype == 'bool' or df[column].dtype == 'object':
      5     df[column].fillna(df[column].mode()[0], inplace=True)
      6 print(f'runtime : {time.time() - s_time}')

     runtime : 1.1033070087432861
```

```
      1 df.shape

     (239062, 49)
```

```
[49]  1 df = df_original
      2 df.shape

     (268850, 49)
```

نال ها را با میانگین و مد پر می کنیم و تایم را مشاهده می کنیم.

سپس همین کار را برای ddf میکنیم. ران تایم کاهش یافت.

```
      1 s_time = time.time()
      2 ddf.fillna(ddf._get_numeric_data().mean())
      3 for column in ddf.columns:
      4   if ddf[column].dtype == 'bool' or ddf[column].dtype == 'object':
      5     ddf[column].fillna(ddf[column].mode()[0])
      6 print(f'runtime : {time.time() - s_time}')

     runtime : 0.1417064666748047
```

البته این قسمت ایراد جزئی دارد که باید رفع گردد.

حال به سراغ outlier Detection با استفاده از ZScore می رویم.

```
[61]  1 def myzscore(df):
      2   for col in df.columns:
      3     if df[col].dtype == 'int64' or df[col].dtype == 'float64':
      4       upper_range = df[col].mean() + 3 * df[col].std()
      5       lower_range = df[col].mean() - 3 * df[col].std()
      6       indexs = df[(df[col] > upper_range) | (df[col] < lower_range)].index
      7       df.drop(indexs)
```

```
[63]  1 s_time = time.time()
      2 myzscore(df)
      3 print(f'Runtime: {time.time() - s_time}')
```

```
Runtime: 4.432937383651733
```

```
1 s_time = time.time()
2 with Pool(cpu_count()) as p:
3   p.imap(myzscore, [ddf.partitions[0], ddf.partitions[1], ddf.partitions[2],
4                     ddf.partitions[3]])
5 print(f'Runtime: {time.time() - s_time}')
```

```
Runtime: 0.251492977142334
```

و میبینیم که ران تایم کاهش داشت.