محمد زیاری - 97222047

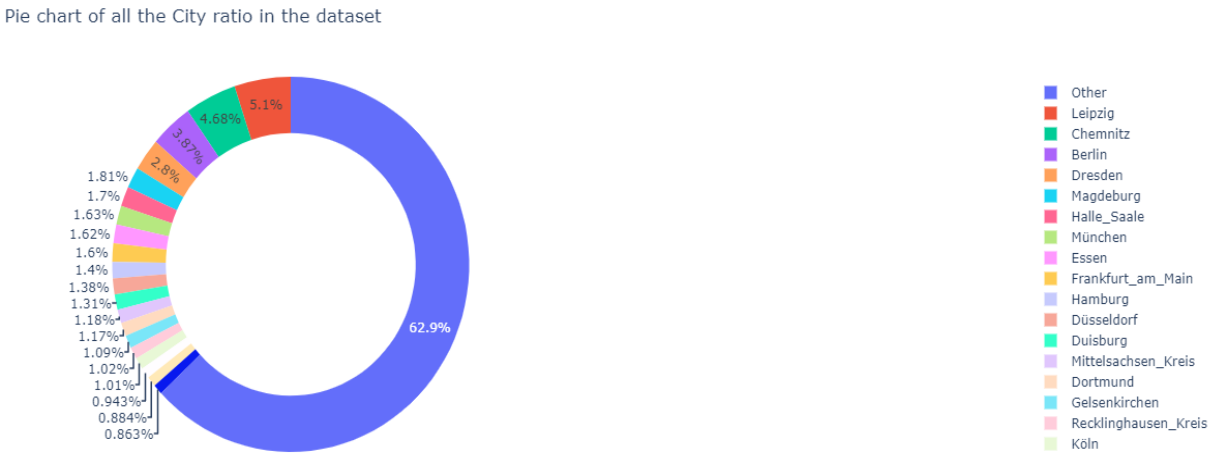
بخش دوم - دیتاست املاک آلمان

- **پاکسازی و مصورسازی داده ها**

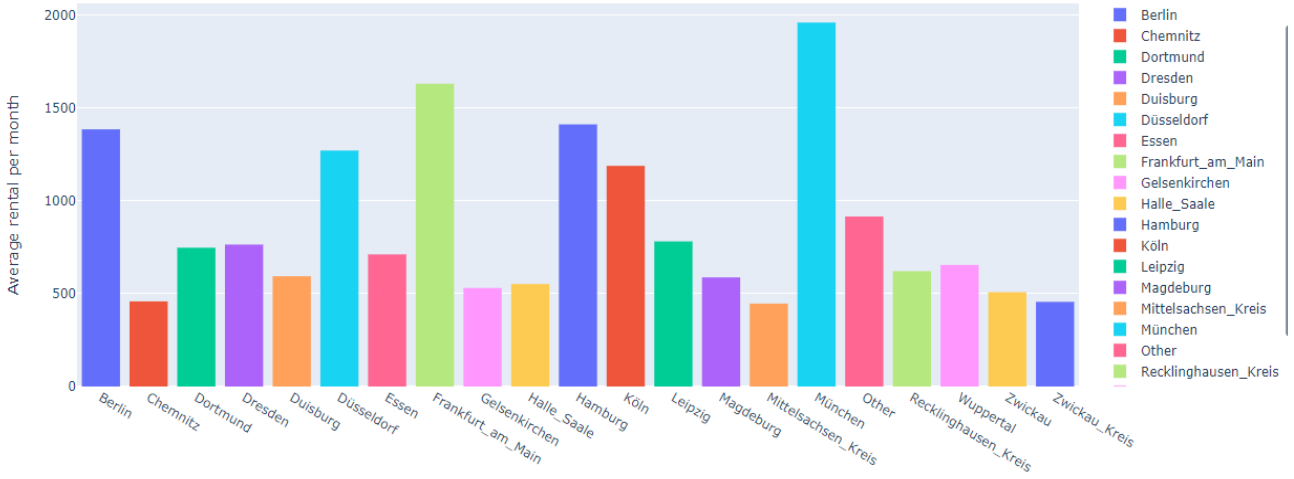
ابتدا داده ها را از ککگل خواندیم تا کارهای مربوط به مرتب سازی داده ها انجام شود. قبل از انجام پاکسازی و پاک کردن بعضی از ستونها چند مرحله مصورسازی و بررسی فیچرها مختلف را در زیر خواهیم داشت.

داده های این املاک در شهرهای مختلف آلمان موجود است و تعداد این شهرها زیاد است. لذا 20 شهری که بیشترین ملک را دارند را جدا کرده و بقیه املاک را با نام others در دیتاست قرار می دهیم.

نمودار پراکندگی تعداد ملک در هر شهر به صورت زیر است که بیشترین مقدار مربوط به شهر لایپزیگ است.

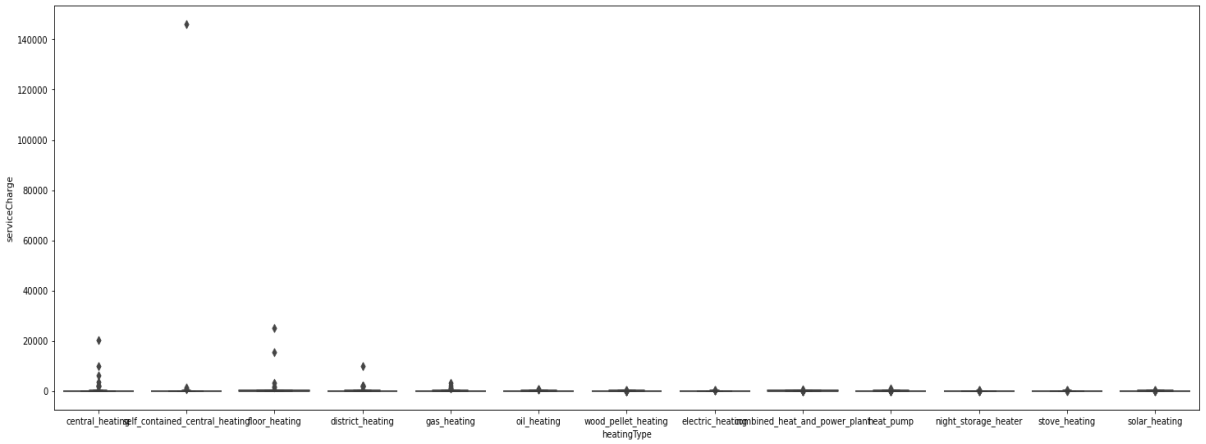


حال در نمودار زیر بررسی می شود که در هر یک از شهرها میزان متوسط اجاره ماهانه به چه شکل می باشد. بیشترین میزان اجاره مربوط به شهرهای مونیخ و فرانکفورت می باشد.



مورد بعدی که بررسی خواهیم کرد نوع خانه و تاثیر آن در اجاره ماهیانه آن می باشد. مشخصا بیشترین نرخ اجاره مربوط به پنت هاوس و کمترین مربوط به واحد های طبقه همکف می باشد.

برای نمودار بعدی به سراغ داده های heatingType و بررسی تأثیر آن بر servicecharge خواهیم رفت. آنچه در نمودار زیر مشخص است آن است که به علت داده های پرت و پراکندگی بالا نمودارها به وضوح و دقت قابل تفسیر نیستند اما به نظر در خانه هایی که گرمایش مرکزی یا زمینی دارند میزان servicecharge اندکی بالاتر می باشد.



حالا که با مشکلات داده های پرت مواجه شدیم به سراغ پاکسازی داده ها خواهیم رفت و در خلال ان چند نمودار دیگر را نیز بررسی می کنیم.

برای پاکسازی داده ها اول از همه بررسی شده است که در هر سطر به چه مقدار دیتای null وجود دارد سپس سطرهایی که داده ی null آن ها بیش از حد است (بیش از نصف) به طوری که اگر با میانگین یا مد پر شود مدل خوبی و پیشبینی خوبی نداشته باشیم را پاک میکنیم که تعداد آنها 7 تا است.

پس از آن 6 ستون که به نظر محتوایی که به پیشبینی ما کمک نمیکنند دارند را از داده ها حذف کردیم.

حالا می دانیم داده های Bool و Object باید عددی شوند تا برای مدل ما قابل فهم باشند. از این جهت ابتدا پراکندگی این داده ها را میسنجیم و داده هایی که پراکندگی بالایی دارند نمیتوانند مدل خوبی تشخیص دهند (حتی اگر one hot encoding کنیم چندین سطر اشغال میکنند) و آنها را حذف کردیم و بقیه داده ها را one hot کردیم.

برای پر کردن داده های null عددی ابتدا از مد استفاده کردم که جواب خوبی نداشت و بعد از آن با همان میانگین جاهای خالی پر شده است.

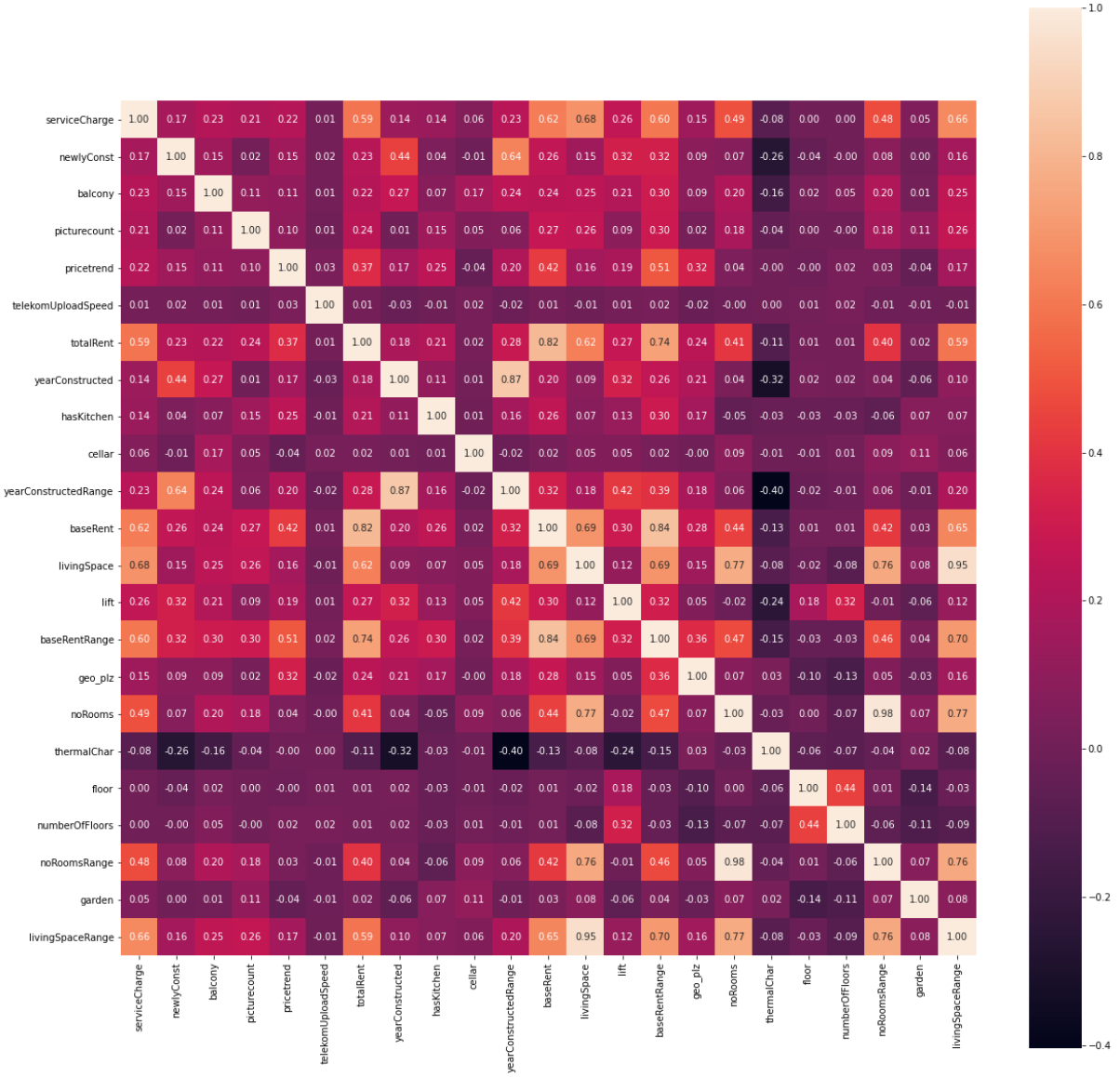
سپس سعی بر آن داریم تا داده های پرت را پاک کنیم. در واقع یک بازه (توزیع نرمال) در نظر گرفتیم و هر داده ای عضو آن بازه نباشد را پاک میکنیم. بعد از آن داده ها را به صورت دستی scale کردیم.

دو معادله برای scale کردن داشتیم، ابتدا معادله اولی که min و max داده را در نظر میگرفت امتحان

کردیم، اما نتیجه خوبی نداشت سپس به سراغ راه دوم scale کردن ینی همان استفاده از std رفتیم، با این روش بهتر جواب داد

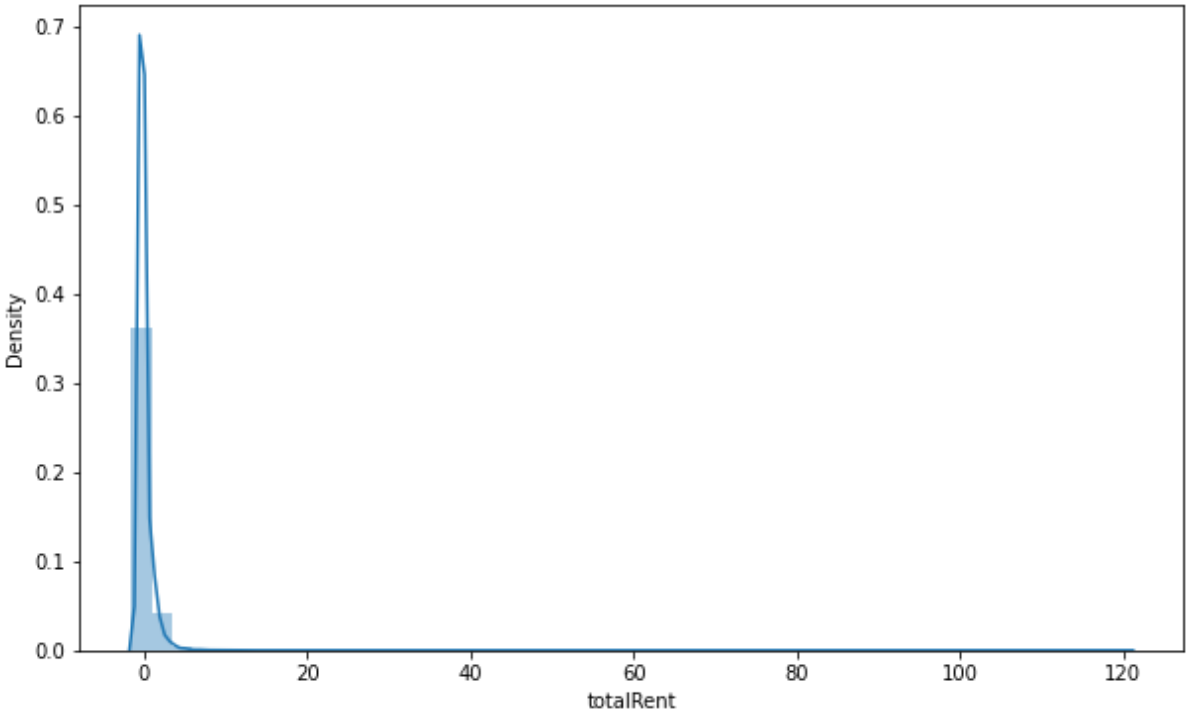
البته از min max scaler هم استفاده کرده بودیم اما در خصوص کورولیشن داده ها و همچنین نتیجه نهایی خوب نبود.

بخش بعدی ماتریس Correlation است که دلیل آنکه one hot بعد آن انجام شده ، آن است که سطر های ماتریس زیاد میشد و تحلیل سخت تر بود.



برای مدلسازی و پیش بینی ما totalRent را هدف قرار دادیم و همانطور که در هیت مپ بالا مشخص شده است داده baseRent بیشترین میزان کورولیشن خطی را با هدف ما که همان totalRent هست دارد، که دارای کورولیشن 0.82 است. دو فیچر telecomuploadspeed , number of floors کمترین میزان ارتباط که 0.01 است را با هدف ما دارند.

آخرین نموداری که بررسی می کنیم نمودار totalRent است که فیچر target ما نیز محسوب می شود که طبق نموداری که در پایین میبینیم از توزیع نرمال پیروی می کند.



پیش از آنکه به سراغ مدل ها و تفسیرشان برویم مواردی که در پاکسازی داده ها به ما کمک خواهند کرد را بررسی می کنیم.

استفاده از فرایند multiprocessing در بخش پاکسازی داده ها به ما کمک میکند تا برخی فرایندهایی که در طی اینکار انجام میدهیم، با سرعت بیشتری انجام شوند. اگر فرایند های ما مستقل از هم باشند و با هم ارتباطی نداشته باشند، میتوانیم در آنجا از multiprocessing استفاده کنیم. معمولا در زمانی که از حلقه ها استفاده میکنیم فرایندی که انجام میشود (به خصوص وقتی حجم داده ها زیاد باشد) زمان بر

خواهد بود، در این حالت موازی سازی فرایند ها میتواند سرعت عملیات ما را بالا ببرد و در زمان کمتری انجام خواهد شد.

در فرایند پاکسازی داده هایی که در این بخش با آنها کار کردیم، در قسمتی Outlier ها را حذف میکنیم، این بخش را میخواهیم یک بار هم به صورت موازی انجام دهیم تا ببینیم runtime چقدر تغییر خواهد کرد.

بدین ترتیب یک تابع به نام outlier تعریف کردیم که در آن عملیات حذف داده های پرت را انجام میدهیم. در این تابع داده هایی که از mean+3*std بیشتر یا از mean-3*std کمتر باشند را از دیتاست حذف میکنیم.

همانطور که مشخص است این عملیات در یک حلقه for انجام میشود و کل دیتاست را میگردد.

کل دیتاست را بر اساس یکی از ستون ها(در اینجا regio1) گروه بندی می کنیم. با این کار دیتاست به چندین گروه تقسیم می شود و با یک for روی تمامی گروه ها عملیات را انجام میدهد.

حال این عملیات را یک بار بدون موازی سازی امتحان میکنیم و مشاهده میکنیم که runtime در این مرحله برابر با 3.873317241668701 میشود.

سپس با استفاده از موازی سازی نیز یک بار دیگر امتحان میکنیم. برای انجام multiprocessing در کولب تنها 2 cpu در اختیار ما قرار داده میشود. میبینیم که runtime در این مرحله برابر با 0.6448357105255127 شده است.

همانطور که مشخص است، فرایند ما سریعتر انجام شده است و علت آن هم همین موازی سازی است که انجام شد.

دو کتابخانه dask و pyspark برای پردازش داده ها حجیم بسیار کاربرد دارند. کتابخانه dask برای انجام پردازش های موازی در پایتون به کار میرود و از pyspark نیز برای کار با داده های large-scale استفاده میشود. در هر دوی آنها می توان دیتا فریم pandas را به مدل خاصی از دیتا فریم که در آنها تعریف شده تبدیل کرد و راحت تر بر روی دیتا ها کار کرد.

با استفاده از آنها، پردازش داده های بزرگ سریعتر انجام خواهد شد و سرعت عملیات بالا خواهد رفت.

- **مدل**

حال که کار ما بر روی داده ها به اتمام رسید نوبت جدا کردن target و feature هاست که به آن ها برچسب x و y زدیم . پس از آن داده ها را با استاندارد اسکیلر اسکیل کردیم و همچنین تمامی مدل ها با استفاده از متدها به 5 بخش 5 بخش و 10 بخش 10 بخش یا همان فولد تقسیم بندی شده اند.

حالا نوبت به پیاده سازی رگرسیون برای داده هاست. ابتدا میانگین خطا یا همان mse را محاسبه میکنیم. توسط پکیج sklearn و با توجه به دستور Linear Regression بر روی داده ها پیاده کرده ایم. این مدل میزان دقتش نزدیک به 71 درصد بود و همچنین میزان خطا یا mse آن نیز حدودا 0.32 بود.

به طور کل میتوان گفت که مدل آماده با استفاده از پکیج از مدل دستی سرعت بیشتری در fit شدن داشت و همچنین از میزان loss پایین تر و به تبع آن مقدار دقت بالاتری برخوردار است هرچند من رگرسیونی که به صورت دستی زده شده بود را از کد پاک کردم.

سپس همان مدل را برای یک فیچر یا ویژگی پیاده سازی میکنیم. که این فیچر همان baseRent است که میزان کولوریشن بیشتری با هدف ما داشت.

میدانیم هرچقدر فیچرهای بیشتری به آن اضافه شوند ، دقت ما بالاتر خواهد بود، که پس از پیاده سازی و اضافه کردن هر فیچر به آن رسیدیم .

همانطور که گفته شد استفاده از یک فیچر برای فیت کردن مدل را انجام دادیم و می بینیم که نتیجه و دقت حاصل کمتر از حالت تمامی فیچر ها است و به این دلیل میتواند باشد که افزایش فیچر ها باعث میشود تا اطلاعات بیشتری داشته باشیم و آن اطلاعات به تخمین زدن ما کمک می کند و تنها یک فیچر ممکن است که نتیجه خوبی نداشته باشد. میزان دقت در این روش 68 درصد بود که به نسبت مدل قبل خطای بیشتری است.

در آخر هم مدل های lasso و ridge را پیاده سازی کردیم که اولی با لرنینگ ریت 0.0001 و دومی با 0.1 میزان دقت مشابه با رگرسیون خطی داشتند و در 10 فولد دقتی 70 درصدی داشت و میزان خطای mse آن نیز با رگرسیون خطی برابر بود.