گزارش پروژه شماره ۱ درس داده‌کاوی دکتر فراهانی تقدیم می‌گردد. لازم به ذکر است که نوت‌بوک‌های ارایه شده، حاوی مارک آپ و توضیحات مجزا برای هر بخش از کدها هستند، به طوری که این توضیحات به تنهایی خلاصه‌ای از کار را شرح می‌دهند، این گزارش برای توضیحات تکمیلی و عمیق‌تر ارایه می‌شود.

دیتاست شماره ۱

در این تمرین از دیتاست Airbnb در شهر نیویورک سال ۲۰۱۹ استفاده شده است. کد این تمرین در نوت‌بوک NYC-AirBNB.ipynb قرار دارد. این دیتاست شامل ۴۸۸۹۴ نمونه و ۱۶ ویژگی است. هنگام لود کردن دیتاست، ستون last_review را به عنوان تاریخ پارس می‌کنم. پس از بررسی اجمالی تعداد ستونها، نوع آن‌ها و ویژگی‌های آماری، تعداد داده‌های گمشده در هر ستون را بررسی می‌کنم. به نظر می‌رسد که در این دیتاست تعداد زیادی از ویژگی‌ها بدون داده گمشده هستند. بجز ویژگی هایی چون host_name، name، last_review و reviews_per_month. ستون last_review که شامل تاریخ آخرین کامنت درباره آگهی است را با مهندسی ویژگی به سه ستون سال، ماه و روز تبدیل می‌کنم، همچنین تایپ آن‌ها را به int تغییر داده و مقادیر گمشده را از روی مد پر می‌کنم (برای روز از روی میانگین=۱۵). حال ستون price که قصد مدلسازی آن را داریم بررسی می‌کنم. این ستون دارای میانگین ۱۵۲ و انحراف معیار ۲۴۰ است. ۱۱ نمونه مقدار ۰ برای price دارند که بی معنی است و چون ۱۱ تعداد کمی است، این نمونه‌ها را حذف می‌کنم. با بررسی ستون‌های host_id و host_name به این نتیجه می‌رسم که بسیاری از نام‌های آگهی‌کنندگان تکراری است، درنتیجه ستون host_id معیار بهتری برای تمایز آگهی‌دهندگان است. در اینجا ستونهای id، name، host_name و last_review را حذف می‌کنم. (آخری را به سال و ماه و روز تبدیل کردم). حالا تنها ستونی که هنوز مقادیر گمشده دارد reviews_pre_month است که مقادیر گمشده آن را با ۰ پر می‌کنیم، چون نمونه‌هایی که این مقدار برای آنها گم شده است، مقدار number_of_reviews آنها هم صفر است. حال با روش z-score داده‌های پرت را حذف می‌کنم. در این مرحله ۴۷۷۲ نمونه حذف می‌شوند. حال در بعضی ویژگی‌ها عمیق‌تر می‌شویم. برای مثال ویژگی neighbourhood_group دارای ۵ مقدار منحصر به فرد است که نشان‌دهنده ناحیه آگهی می‌باشد. مناطق Brooklyn و Manhattan بیشترین تعداد آگهی را دارا می‌باشند و حدود ۸۸٪ آگهی‌ها را در برمی‌گیرند. پس می‌توانیم سایر مقادیر را به other نظیر کنیم. این کار باعث کاهش ویژگی‌ها پس از one-hot encoding می‌شود. ویژگی neighbourhood شامل محله‌های جزئی تر می‌شود و حدود ۱۶۸ مقدار منحصر به فرد دارد، از آنجایی که مقادیر این ستون اختلاف زیادی با هم ندارند، نمی‌توان همانند ستون قبلی مقادیری را به others نظیر کرد. پس از one-hot encoding این ستون ۱۶۸ ویژگی به فضای ویژگی ما اضافه می‌کند پس بهتر است که از آن صرف‌نظر کنیم. از ستون room_type درمی‌یابیم که بیشتر آگهی‌ها یا دارای اتاق اختصاصی هستند و یا کل خانه در اختیار است و آگهی‌های کمتری به اتاق اشتراکی اختصاص دارد. همچنین به نظر می‌رسد که آگهی‌های منطقه Manhattan، رنج قیمتی بالاتری دارند. فعال‌ترین آگهی‌کننده، حدود ۱۰۳ آگهی ثبت کرده است، اما سوالی

که مطرح شده این است که این افراد، معمولا چند خانه را آگهی کرده‌اند (ممکن است یک خانه چند بار آگهی شده باشد). برای نفر اول فعال‌ترین‌ها، تعداد خانه‌های منحصر به فرد ۱۰۱ است، این مقدار از روی latitude های منحصر به فرد بدست آمده است. همچنین در برخی از سایر آگهی کنندگان نیز این اختلاف بین تعداد آگهی و تعداد خانه‌های منحصر به فرد وجود دارد که تئوری وجود چند آگهی برای یک خانه را تایید می‌کند (شاید در بازه‌های زمانی مختلف). برخی از فعال‌ترین آگهی‌کنندگان، در بیش از ۱۰ neighbourhood صاحب خانه هستند. اکثر این افراد در نواحی Manhattan، Brooklyn و Queens خانه آگهی کرده‌اند و اثری از دو منطقه دیگر (Bronx, Staten Island) دیده نمی‌شود. اگر تعداد کامنت برای یک آگهی را بتوانیم شاخصی از تعداد مشتریان یا محبوبیت آگهی در نظر بگیریم، لیستی از صاحبان آگهی که بیشترین مشتری را دارند ارایه شده است. تعداد آگهی برای ۱۰ شخص برتر این لیست نشان داده شده است و صاحب آگهی با host_id شماره 6885157 بیشترین تعداد مشتری یعنی ۱۱۸۴ را داشته است. در ماتریس همبستگی می‌بینیم که number_of_reviews پس از reviews_per_month، بیشترین همبستگی را با last_review_year، availability_365 و minimum_nights دارد. پس اگر یک آگهی به اندازه کافی جدید باشد، در بیشتر روزهای سال در دسترس باشد و حداقل تعداد شب‌هایی که خانه اجاره داده می‌شود کمتر باشد، کامنت‌های بیشتر و در نتیجه مشتریان بیشتری خواهد داشت.

در ادامه چند تست فرض مطرح می‌شود:

- آیا price از توزیع نرمال پیروی می‌کند؟ خیر
- آیا میانگین price برابر ۱۵۰ است؟ خیر، ۱۳۸
- آیا reviews_per_month و number_of_reviews توزیع آماری مشابهی دارند؟ خیر
- آیا reviews_per_month و number_of_reviews دارای همبستگی هستند؟ بله
- آیا availability_365 و number_of_reviews دارای همبستگی هستند؟ بله
- آیا calculated_host_listings_count و number_of_reviews دارای همبستگی هستند؟ بله

پس در ادامه بحث قبلی، اگر صاحب آگهی تعداد آگهی‌های زیادی داشته باشد (در سایت فعال باشد)، به احتمال زیاد آگهی‌های آینده او هم دارای کامنت بیشتر و درنتیجه مشتریان بیشتری خواهند بود.

در بخش آخر پیش‌پردازش نهایی را انجام داده و یک مدل رگرسیونی می‌سازم. ابتدا بیایید درباره یک ویژگی جدید بحث کنیم. فاصله یک خانه از مراکز مهم شهری، می‌تواند ویژگی تاثیرگذاری در قیمت و تعداد مشتریان آن خانه باشد. ما طول و عرض جغرافیایی هر خانه را داریم. فرمولی به نام هاورسین وجود دارد که با دریافت یک جفت طول و عرض جغرافیایی از دو نقطه روی کره زمین، فاصله آن دو نقطه را به کیلومتر به‌دست می‌دهد. من از این فرمول استفاده کردم و یک ویژگی جدید با نام dist_jkf که بیانگر فاصله خانه

از فرودگاه بین‌المللی جان‌اف‌کندی است را به دیتاست اضافه کردم. ویژگی‌های دیگری نیز به همین ترتیب می‌توان اضافه کرد (با مراکز مهم دیگر نیویورک) اما من فعلا به همین ویژگی بسنده می‌کنم.

حال ویژگی‌های host_id، neighbourhood، last_review_day، longitude و latitude را حذف می‌کنم. تنها ویژگی‌های غیرعددی باقیمانده، neighbourhood_group و room_type هستند که آن‌ها را با one-hot encoding عددی کرده و سپس آنها را نیز حذف می‌کنم. در آخر تنها ۱۵ ویژگی باقی می‌ماند که بنظرم نیازی به استفاده از PCA نیست.

حال ۱۰٪ از داده‌ها را با train_test_split به عنوان داده تست جدا می‌کنم. سپس ویژگی‌ها (x) را با StandardScaler و خروجی (y) را با روش MinMax (بصورت دستی) نرمال می‌کنم. دلیل عدم استفاده از StandardScaler برای y این است که ممکن است برخی مقادیر قیمت را منفی کند که زیاد منطقی نیست. نکته حایز اهمیت در هنگام نرمالسازی این است که باید نرمال‌ساز را ابتدا روی داده‌های آموزش fit کرده و سپس روی داده‌های آموزش و تست اعمال کنم، چون در دنیای واقعی فرض بر این است که به داده‌های تست و در نتیجه ویژگی‌های آماری آن‌ها دسترسی نداریم. برای آموزش از مدل RandomForestRegressor استفاده می‌کنم، چون یک مدل Ensemble است و مدل‌های مختلفی را آموزش داده و خروجی را از نتیجه آن‌ها محاسبه می‌کند. این مدل از رگرسیون خطی ساده LinearRegression بهتر عمل می‌کند و خطای میانگین مربعات ۰.۰۰۸۷ روی داده‌های تست نتیجه می‌دهد که در مقایسه با مقدار ۰.۰۱ رگرسیون خطی بسیار کمتر است. اما در مقابل RandomForest زمان بیشتری را صرف آموزش می‌کند و کندتر است.

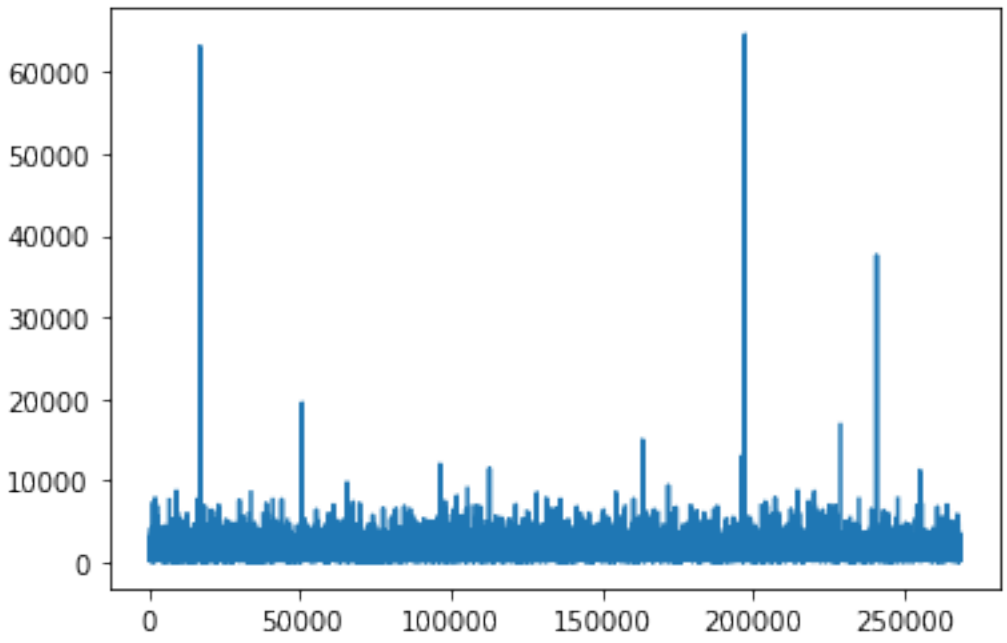
دیتاست شماره ۲

در این تمرین از یک دیتاست استخراج شده از یکی از سایت‌های املاک کشور آلمان استفاده می‌شود که اطلاعات مربوط به اجاره یک واحد را در بردارد. کدهای مربوط به این تمرین در نوت‌بوک Apartment.ipynb قرار دارند. این دیتاست شامل ۲۶۸۸۵۰ نمونه و ۴۹ ویژگی است. پس از بررسی ستون‌ها و نوع‌داده‌ای آنها و معیارهای آماری، اولین کاری که انجام می‌دهم لیست کردن ستون‌های غیرعددی به همراه تعداد مقادیر منحصر به فرد برای هر ستون است. در اینجا ستون‌هایی که مقادیر منحصر به فرد زیادی دارند را پس از بررسی اولیه حذف می‌کنیم. در این مرحله ۹ ویژگی را حذف می‌کنم. ویژگی‌های دیگری نیز وجود دارند که مقادیر منحصر به فرد زیادی دارند، اما در ادامه به آن‌ها نیاز پیدا می‌کنم، پس دیرتر حذفشان می‌کنم.

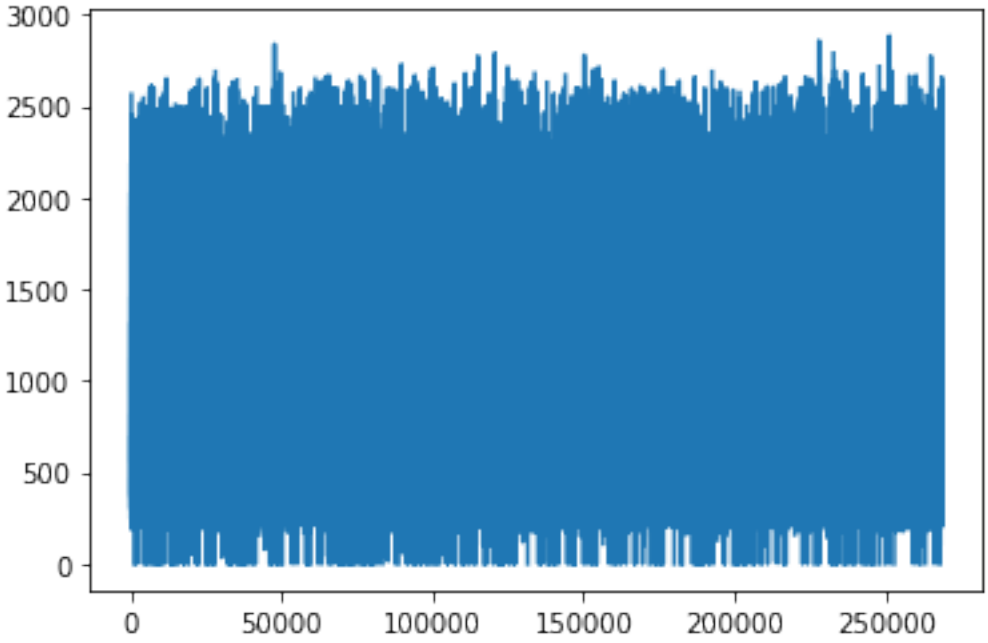
ویژگی	توضیحات	تعداد مقادیر منحصر به فرد
date	تاریخ استخراج داده	۴
facilities	امکانات (به زبان آلمانی)	۱۸۹۵۲۶
description	توضیحات آگهی (به زبان آلمانی)	۲۱۲۶۲۱
streetPlain	نام خیابان با فرمت متفاوت	۵۴۴۹۰
street	نام خیابان	۵۲۳۷۳
houseNumber	پلاک منزل	۵۵۱۰
geo_bln	استان (با ویژگی دیگری یکسان است)	۱۶
scoutId	شناسه فرد در سایت	۲۶۸۸۵۰
firingTypes	منبع انرژی اصلی	۱۳۲

پس از این مرحله تعداد ویژگی‌ها به ۴۰ می‌رسد. در مرحله بعد، باید در صورت وجود داده پرت، آن‌ها را حذف کنیم. به طور مثال بیایید ستون totalRent را بررسی کنیم. این ستون دارای میانگین ۹۰۰، انحراف معیار حدود ۳۳۰۰۰ و بیشترین مقدار ۱۵۰۰۰۰۰ است، پس به نظر داده‌های پرت در دیتاست وجود دارند. پس از روش z-score برای حذف داده‌های پرت استفاده می‌کنم (داده‌هایی که اندازه z-score آن‌ها از ۳ بیشتر است را دور میریزم).

در این مرحله حدود ۱۳۱۳۲ نمونه حذف می‌شوند. اما با بررسی مجدد totalRent به این نتیجه رسیدم که هنوز تعدادی از این داده‌های پرت باقی‌مانده اند. در نمودار زیر می‌بینید:



یک بار دیگر متد قبلی را با مقدار ۴ تکرار می‌کنم:



مشاهده می‌شود که داده‌های پرت حذف شدند. پس از این مرحله نیز ۷۹۹۲ نمونه حذف می‌شوند. اگر در مرحله دوم به جای عدد ۴ از ۳ استفاده می‌کردم، ۵۰۰۰ نمونه بیشتر حذف می‌شدند. پس نهایتا پس از حذف داده‌های پرت، ۲۴۷۷۲۶ داده باقی مانده‌اند.

در مرحله بعدی، داده‌های گمشده را بررسی می‌کنم. یک دیتافریم جدید از ویژگی‌هایی که داده گمشده دارند به همراه درصد مقادیر گمشده و همچنین نوع داده‌ای آن‌ها برای بررسی بهتر موقعیت می‌سازم. ابتدا ویژگی‌هایی که بیش از ۵۰٪ مقادیر آن‌ها گم‌شده است را حذف می‌کنم، بجز electricityBasePrice و heatingCosts (در ادامه مشخص می‌شود چرا)، در اینجا ویژگی‌ها از ۴۰ به ۳۵ می‌رسند.

برخی از نمونه‌ها شامل مقادیر غیرمعتبر برای برخی ویژگی‌ها هستند. به طور مثال مقدار ۰ برای livingSpace، baseRent و totalRent معنایی ندارد. چون دو ویژگی اول فاقد مقادیر گمشده هستند، نمونه‌هایی که مقدار ۰ دارند را حذف میکنیم (کمتر از ۱۵۰ نمونه) اما چون ویژگی آخر شامل مقادیر گمشده است، مقادیر ۰ آن را گمشده در نظر می‌گیریم (۲۲۲ نمونه).

در مرحله بعد با استفاده از متد duplicated و ویژگی‌هایی که مقادیر گمشده ندارند و می‌توانند تمایز بین نمونه‌ها ایجاد کنند، نمونه‌های تکراری را شناسایی کرده و حذف می‌کنم (۲۶۰۳۴ نمونه). پس از این مرحله دیگر به ویژگی‌های regio2، regio3 و geo_krs نیازی ندارم پس آن‌ها را حذف می‌کنم. (۳۲ ویژگی باقی‌مانده)

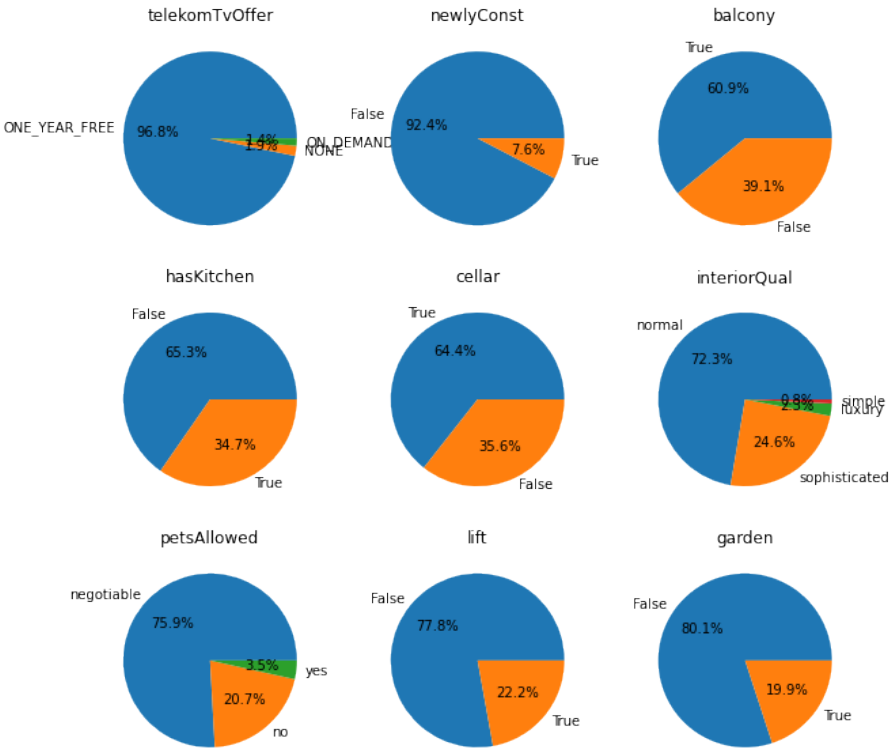
در مرحله بعد سعی در پرکردن مقادیر گمشده داریم. مهم‌ترین نکته این مرحله این است که ویژگی totalRent که به عنوان هدف قرار است روی آن مدلسازی انجام دهم، شامل ۱۵٪ داده گمشده است. برای حل این مشکل می‌توان از راهکارهای مختلفی استفاده کرد. یک راهکار این است که این ۱۵٪ را به عنوان داده تست برداشته و مدلسازی را روی ۸۵٪ باقیمانده انجام دهم. یک راه حل شلخته این است که مقادیر گمشده را با ویژگی‌های آماری مثلا میانگین پر کنم. و راه سوم که من از آن استفاده کردم این است که یک رابطه خطی بین اجاره‌بهای کامل یک واحد ویژگی‌هایی مانند baseRent، serviceCharge، electricityBasePrice و heatingCosts در نظر بگیرم. من از فرمول زیر استفاده می‌کنم که به صورت تجربی به آن رسیده‌ام، قطعا از فرمولهای دیگری نیز می‌توان استفاده کرد:

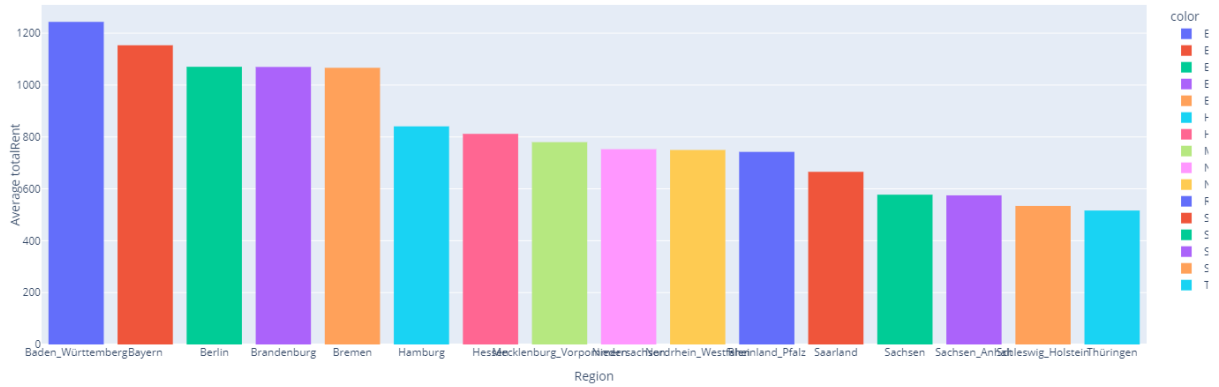
$$\text{totalRent} = \text{baseRent} + \text{serviceCharge} + \alpha \times \text{electricityBasePrice} + \beta \times \text{heatingCosts}$$

ابتدا شهودم برای رسیدن به این فرمول را اندکی شرح می‌دهم. به طور قطع در هزینه اجاره نهایی، هزینه پایه موثر است. من میانگین ستون baseRent را از میانگین ستون baseRent کم کردم، این مقدار باید با ویژگی‌های دیگر پر شود. با بررسی مقادیر میانگین ویژگی‌های دیگر، به این نتیجه رسیدم که مقدار serviceCharge به صورت خطی و دو مقدار دیگر با ضریبی کمتر از ۱ باید به معادله اضافه شوند. اما اینجا یک مشکل وجود دارد و آن این است که ۸۴٪ ستون electricityBasePrice، ۸٪ ستون heatingCosts

و ۲.۵٪ ستون serviceCharge شامل داده‌های گمشده است. ابتدا این داده‌ها را با میانگین‌شان پر می‌کنم و سپس از آنها در فرمول استفاده می‌کنم. حال سوالی که پیش می‌آید این است که چرا این راه از پرکردن با میانگین بهتر است؟ اگر به رابطه دقت کنید، بخش اعظمی از آن را ویژگی baseRent (فاقد مقدار گمشده) و serviceCharge (مقدار گمشده بسیار کم: ۲.۵٪) می‌سازند و بخش کوچکی از آن به دو ویژگی دیگری که مقادیر گمشده زیادی داشتند وابسته است. من به طور شهودی آلفا را $\frac{۱}{۹}$ و بتا را $\frac{۱}{۷}$ در نظر می‌گیرم و مقادیر گمشده totalRent را با آن پر می‌کنم. پس از این مرحله به دو ویژگی electricityBasePrice و heatingCosts دیگر نیازی ندارم پس آن‌ها را حذف می‌کنم. (۳۰ ویژگی باقی مانده) حال در ادامه، مقادیر گمشده ویژگی‌های عددی را با میانگین آن‌ها و مقادیر گمشده ویژگی‌های غیر عددی را با مد آنها پر می‌کنم.

کار را با کمی مصورسازی داده‌ها ادامه می‌دهم و نمودار دایره‌ای برخی ویژگی‌های غیر عددی (که مقادیر منحصر به فرد زیادی ندارند) را رسم کرده و بررسی می‌کنم. چهار ویژگی condition، regio1، heatingType و typeOfFlat مقادیر منحصر به فرد زیادی دارند (بیشتر مساوی ۱۰). نمودار هیستوگرام آن‌ها را رسم می‌کنم. در اینجا مشاهده می‌شود که بیشتر مقادیر هر ویژگی مربوط به یک یا چند دسته‌بندی است، پس سایر دسته‌بندی‌های کم تکرار را به عنوان others در نظر می‌گیرم. این کار به کاهش فضای ویژگی پس از one-hot encoding کمک می‌کند.





Hamburg، Berlin و Bayern است. در ادامه پس از بررسی هیت‌مپ همبستگی به این نتیجه می‌رسم که برخی ویژگی‌ها با همدیگر همبستگی بسیار بالایی هستند و درواقع نشان دهنده یک مفهوم در فرمت‌های متفاوت هستند (ویژگی‌هایی در انتهای نامشان Range آمده، نشان دهنده دهک ویژگی اصلی هستند)، این ویژگی‌ها را حذف می‌کنم (۲۶ ویژگی باقی‌مانده)

در ادامه پیش‌پردازش نهایی و آموزش مدل را دنبال می‌کنم. ویژگی‌هایBoolean را به جای one-hot encode کردن به int تبدیل می‌کنم تا مقدار ۰ و ۱ داشته باشند. اینکار باعث کاهش بعد فضای ویژگی می‌شود. همچنین ویژگی‌های غیرعددی را one-hot encode می‌کنم. درپایان این مرحله تعداد ویژگی‌ها به ۵۳ می‌رسد. با استفاده از PCA این مقدار را به ۲۸ کاهش می‌دهم و پس از نرمالسازی ویژگی‌ها و هدف (x,y) همانند دیتاست قبلی (توضیحات قبلی)، به آموزش مدل می‌پردازم.

در بخش پایانی سعی در موازی‌سازی دو عملی دارم که در طی پروژه انجام شده است. یکی پرکردن مقادیر گمشده ستون‌ها و دیگری پیداکردن و حذف داده‌های پرت هر ستون. به نظر می‌رسد که این دو عمل برای هر ستون به صورت مستقل انجام می‌شوند. من با استفاده از ماژول multiprocessing زمان اجرای این دو عمل را با و بدون موازی‌سازی با هم مقایسه کردم، موازی‌سازی باعث کاهش چشمگیر زمان اجرا شد.

نتایج برای داده‌های پرت و داده‌های گمشده به ترتیب به قرار زیر است:

```
Elapsed time: 2.3919887542724615s.
```

```
Elapsed time: 0.15964579582214355s.
```

```
Elapsed time: 1.5286486148834229s.
```

```
Elapsed time: 0.16181969642639l6s.
```

در ادامه با پکیج dask کار می کنم. ابتدا دیتافریم را لود میکنم بدون اینکه آنرا پارتیشن کنم، هدفم این است که ببینم آیا در یک عمل groupby، پانداز بهتر عمل می کند یا دسک؟ نتایج کمی تکان دهنده است. به ترتیب دسک با compute، دسک بدون compute و پانداز:

```
%%time
monthly_total = ddf.groupby(['regio1'])['totalRent'].mean().compute()
```

```
CPU times: user 117 ms, sys: 4.8 ms, total: 122 ms
Wall time: 167 ms
```

```
[70] %%time
monthly_total = ddf.groupby(['regio1'])['totalRent'].mean()
```

```
CPU times: user 33 ms, sys: 81 µs, total: 33.1 ms
Wall time: 67 ms
```

```
[71] %%time
monthly_total = DF.groupby(['regio1'])['totalRent'].mean()
```

```
CPU times: user 34.9 ms, sys: 1.72 ms, total: 36.6 ms
Wall time: 50.8 ms
```

دسک بدون compute هیچ چیزی را محاسبه نمی‌کند بلکه فقط دستورات لازم را ارزیابی کرده و در حافظه ذخیره می‌کند تا در صورت فراخوانی compute آن‌ها را اجرا کند. علت این که دسک با compute کندتر است در داکیومنتیشن آن ارایه شده است:

Best Practices

It is easy to get started with Dask DataFrame, but using it *well* does require some experience. This page contains suggestions for best practices, and includes solutions to common problems.

Use Pandas

For data that fits into RAM, Pandas can often be faster and easier to use than Dask DataFrame. While "Big Data" tools can be exciting, they are almost always worse than normal data tools while those remain appropriate.

در ادامه با دو پارتیشن دسک، یک متد (مثلا یافتن داده‌های پرت) را موازی‌سازی می‌کنیم. که این از حالتی که کل داده را یکجا با پانداز استفاده کنیم سریعتر است.

```
[74] s_time = time.time()
     outlier_delete_whole(DF)
     print(f'Elapsed time: {time.time()-s_time}s.')

     Elapsed time: 5.5979719161987305s.

[75] s_time = time.time()
     with Pool(cpu_count()) as p:
       p.imap(outlier_delete_whole, [ddf.partitions[0], ddf.partitions[1]])
     print(f'Elapsed time: {time.time()-s_time}s.')

     Elapsed time: 0.22002625465393066s.
```