Data Science

Gatherer: Mahdi Momeni

Email: momeni.mpost@gmail.com

Telegram: @master_prg

Linkedin: mehdee81

Github: mehdee81

Github Repo: https://github.com/mehdee81/Data-Science

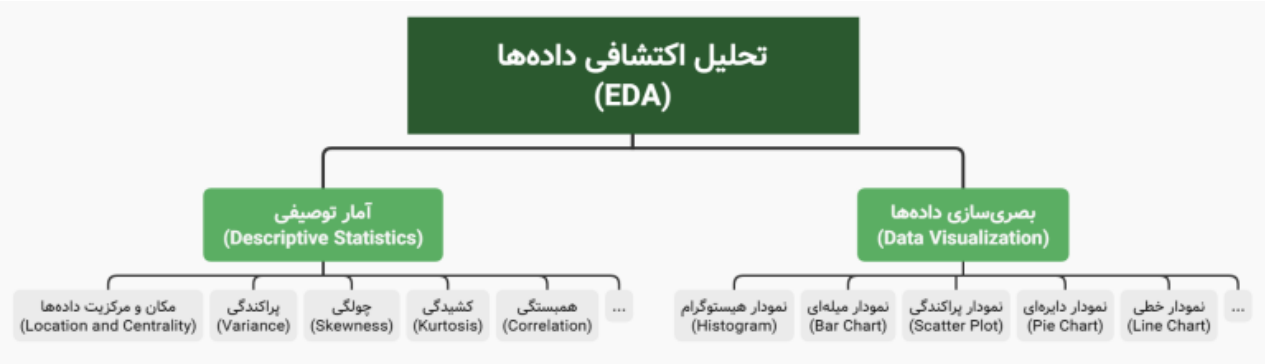
فهرست مطالب

1. تجزیه و تحلیل اکتشافی داده ها Exploratory data analysis (EDA)
2. کتابخانه Numpy
3. کتابخانه Pandas
4. کتابخانه Matplotlib
5. کتابخانه Seaborn
6. تحلیل توصیفی
7. تحلیل توزیع داده ها
8. تحلیل همبستگی
9. آزمون فرضیه
10. یادگیری ماشین
11. یادگیری نظارت شده
    a. رگرسیون (Regression)
    b. طبقه بندی (Classification)
12. بهینه سازی (Hyperparameter optimization)
13. یادگیری بدون نظارت
    a. خوشه بندی (Clustering)
        i. Kmeans
        ii. خوشه بندی سلسه مراتبی (Hierarchical clustering)
14. کاهش ابعاد یا Dimensionality reduction
15. تحلیل مولفه اصلی یا Principal component analysis (PCA)
16. سیستم های پیشنهاد دهنده یا Recommender systems
    a. Content Based
    b. Collaborative Filtering
17. Regex
18. اصول پاکسازی داده (Principles of data cleaning)
19. اصول مستندسازی و گزارش نویسی

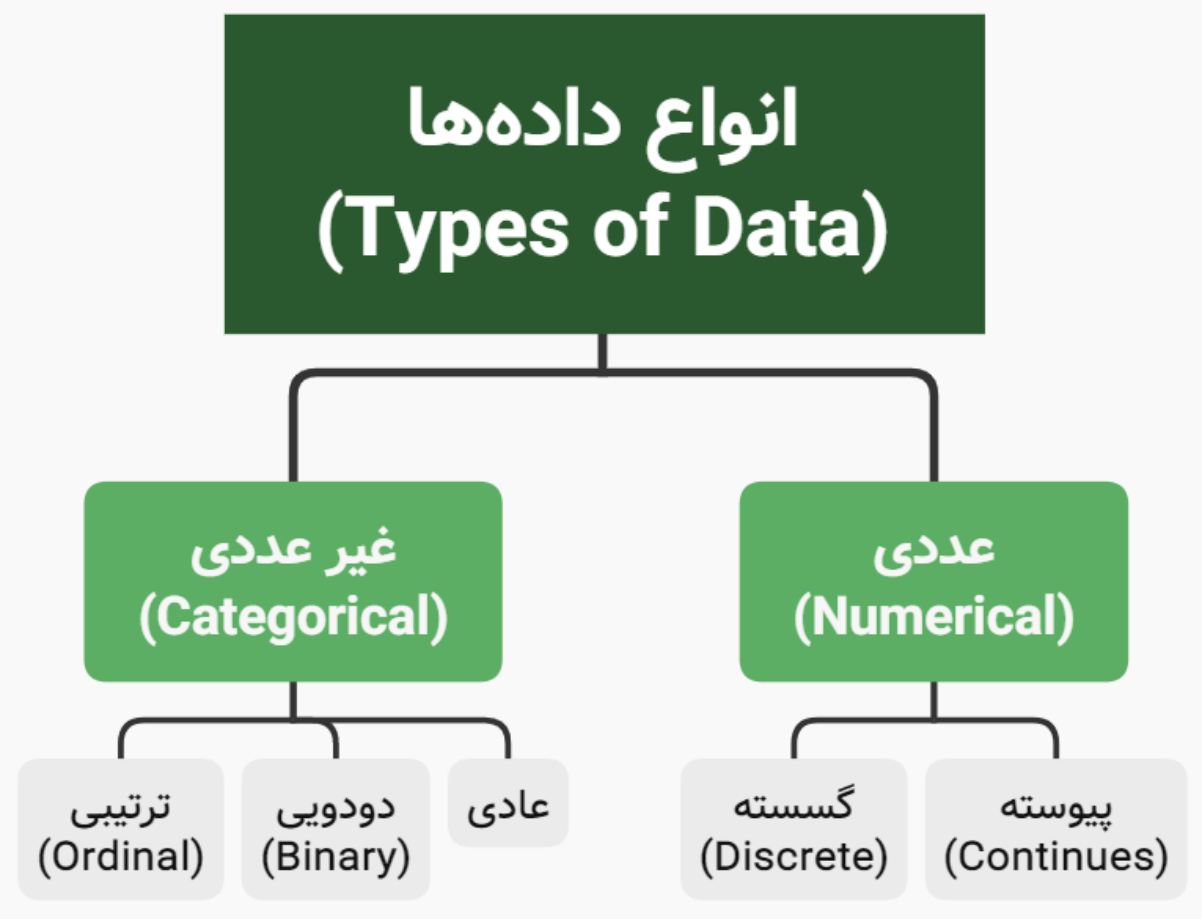
EDA (Exploratory Data Analysis)

به کمک EDA و تکنیک‌ها و مفاهیم آن شما می‌توانید داده‌ها را خلاصه سازی کنید. یک شمای کلی از داده‌ها به افراد مختلف نمایش دهید و یا با انواع نمودارها و گراف‌ها، خلاصه‌ای از داده‌ها را به افرادی که به آن‌ها نیاز دارند نمایش دهید. در واقع EDA را می‌توان از جنبه‌ای اولین قدم برای رسیدن به یک تحیل یا همان آنالیز داده‌ها و یک شمای کلی از داده‌ها دانست.

با استفاده از تکنیک‌ها و روش‌های موجود در EDA، خواهید توانست داده‌ها را ساده‌تر در ذهن خود جای دهید و تحلیل و بررسی کنید.

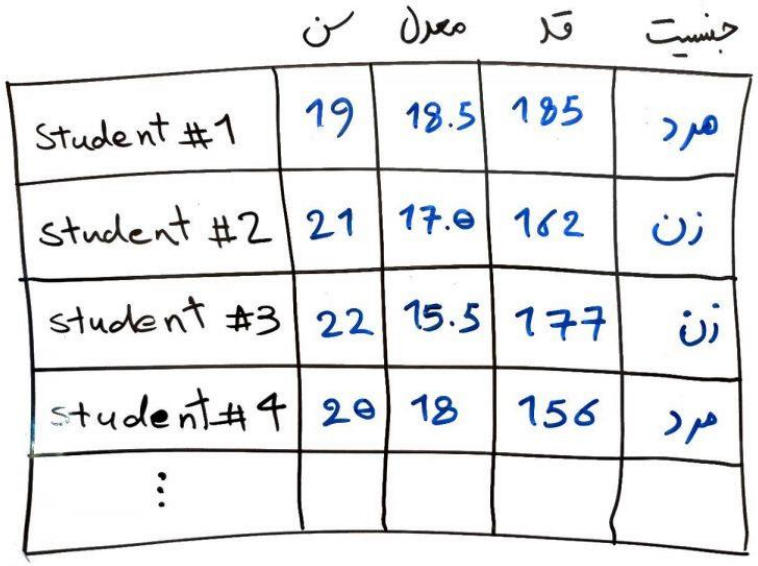


در داده‌کاوی و آمار داده‌ها به صورت‌ها مختلف تقسیم بندی می‌شوند. متخصصان علم داده با توجه به کاربرد و کارکرد داده‌ها، آن‌ها را به دسته‌های مختلفی تقسیم بندی کرده‌اند که هر دسته ویژگی‌های خاص خود را دارد. در کل داده‌ها در علم آمار و داده‌کاوی به دو دسته عددی (Numeric) و غیرعددی (Categorical) تقسیم بندی می‌شوند. هر کدام از آن‌ها به دسته‌های مختلف تقسیم بندی می‌شوند. نگاهی به شکل زیر بی اندازید:



داده های مستطیلی (Rectangular Data)

در داده‌کاوی معمولا سعی می‌شود که داده‌ها به صورت مستطیلی یا همان Rectangular شکل بگیرد تا بتوانند عملیات هایی مانند طبقه‌بندی یا خوشه‌بندی و .... را بر روی آن‌ها انجام دهند.



داده پرت (Outlier) در داده‌کاوی

داده‌های پرت در بعضی مواقع می‌توانند دردسر ساز باشند و در بعضی مواقع هم خود مسئله تشخیص داده‌های پرت به نوعی تشخیص ناهنجاری (Anomaly Detection) است که در آن ما به دنبال یافتن داده‌هایی هستیم که پرت هستند.

فرض کنید می‌گویند میانگین حقوق در یک شرکت ۴میلیون تومان است. آیا این بدان معنی است که اکثر افراد حاضر در آن شرکت ۴میلیون تومان (یا نزدیک به آن) حقوق می‌گیرند. در نگاه یک غیرمتخصص بلی ولی در نگاه یک متخصص آمار و داده‌کاوی قطعا جواب خیر است. ممکن است ۹۵درصد افراد حاضر در آن شرکت حقوق ۱میلیون تومان بگیرند و ۵درصد بقیه حوقشان ۲۰میلیون تومان باشد. در واقع این ۵درصد نوعی داده پرت هستند که میانگین (Mean) را به نفع خود جا به جا کرده‌اند!

تخمین مکان داده‌ها (Estimation Of Location)

اگر یک نفر از شما بخواهد بپرسد که میانگین سن افراد ورودی به دانشگاه شما چه سنی است احتمالا خیلی سریع پاسخ می‌دهید ۱۸سال. ولی اگر یک نفر از شما بپرسد که میانگین حدودی سن خانواده درجه ۱شما چند سال است قطعا جواب نه راحت خواهد بود و نه دقیق. زیرا برای مثال پدر شما ۵۰سال دارد و خود شما ۲۳سال و برادر کوچکتر شما ۱۵سال دارد. به این ترتیب اختلاف سنی در خانواده شما بسیار بیشتر از ورودی‌های یک دانشگاه خاص است. اینجاست که مبحث تخمین مکان داده‌ها یا همان Estimation Of Location معنا پیدا می‌کند تا یک متخصص علوم داده یا مهندس آمار بتواند یک تخمین درست و ساده از داده‌ها داشته باشد.

میانگین (Mean)

جمع تمامی مقادیر و تقسیم نتیجه بر تعداد کل داده ها

میانگین وزن دار (Weighted Mean)

اگر هر کدام از داده‌های ما یک وزن مشخص داشته باشند و بخواهیم میانگین آن‌ها را حساب کنیم ابتدا باید این داده‌ها را در وزن آن‌ها ضرب کنیم. سپس با هم جمع کرده و تقسیم بر تعداد ضرایب کنیم. این کار را در کارنامه یک ترم دانشگاهی خود حتما دیده‌اید. مثلا درس تربیت بدنی ضریب 2 دارد و درس آمار و احتمالات ضریب ۳ و به همین ترتیب بقیه دروس هر کدام ضریب خود را دارند. حالا وقتی می‌خواهند معدل یک ترم شما را حساب کنند، هر کدام از دروس را در ضریب (یا همان وزن آن) ضرب می‌کنند و سپس این مقادیر را با هم جمع کرده و تقسیم بر تعداد ضرایب می‌کنند تا معدل ترم شما حساب شود.

مُد (Mode)

مقداری که بیشترین تکرار را در میان داده‌ها دارد. برای مثال فرض کنید سن افراد حاضر در یک کلاس به صورت زیر است:

۱۸, ۱۹, ۱۸, ۱۸, ۲۰, ۶۰, ۲۱, ۲۰, ۱۸, ۱۹

همان طور که می‌بینید بیشترین تکرار را عدد ۱۸ با 4 بار تکرار داشته است. پس مُد برای این داده‌ها عدد ۱۸ است.

میانه (Median)

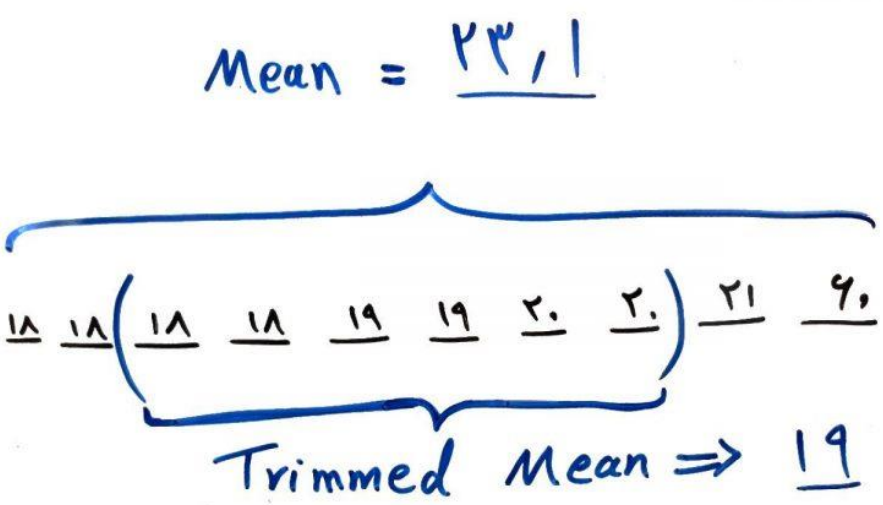
ابتدا داده‌هایی را که دارید به ترتیب مرتب کنید. سپس مقدار وسطی (که نصف داده‌ها از آن بیشتر باشند و نصف داده‌ها از آن کمتر باشند) را انتخاب کنید. این مقدار همان مقدار Median است.

۱۸, ۱۸, ۱۸, ۱۸, ۱۹, ۱۹, ۲۰, ۲۰, ۲۱, ۶۰

مقدار Median برای داده‌های فوق برابر ۱۹ است.

میانگین برش‌خورده (Trimmed Mean)

داده‌ها را به ترتیب بچینید. N درصد از بالای داده‌ها و n درصد از پایین داده‌ها را بردارید (برش دهید). حال میانگین مقادیر باقی مانده را بگیرید. به این کار به اصلاح Trimmed Mean می‌گویند. شکل زیر را ببینید

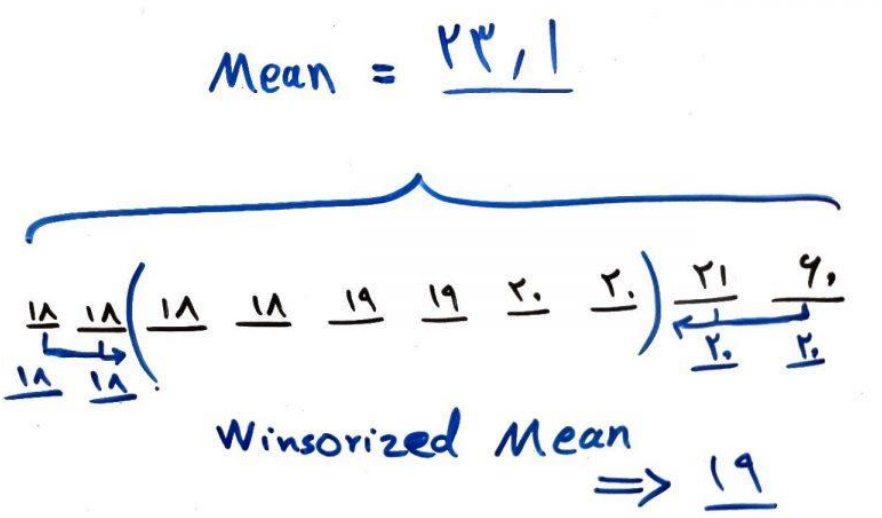


میانه-میانگین (Mid-Mean)

مانند میانگین برش‌خورده (Trimmed Mean) است با این تفاوت که ۲۵درصد از بالای داده‌ها و ۲۵درصد از پایین داده‌ها را برمی‌داریم و از میان داده‌های باقی مانده در بین این دو (۲۵ تا ۷۵درصد) میانگین را محاسبه می‌کنیم. با این کار باز هم میانگین محاسبه شده نسبت به داده‌های پرت قوی (Robust) است.

میانگین Winsorized

این میانگین هم مانند میانگین برش خورده (Trimmed Mean) است با این تفاوت که داده‌هایی که از بالا و پایین قرار است حذف شوند، حذف نمی‌شوند. این داده‌ها تبدیل به بیشتر و کمتر مقدار باقی مانده شده و در محاسبه میانگین حساب می‌شوند. شکل زیر را نگاه کنید



میانه بازه (Mid-Range)

کوچک‌ترین مقدار و بزرگترین مقدار را با هم جمع و سپس تقسیم بر ۲ می‌کنیم. این میانگین شدیدا به داده‌های پرت دار حساس و در واقع قدرت کمی نسبت به داده‌های پرت دارد.

تخمین تنوع و پراکندگی (Estimation Of Variability)

واریانس: واریانس به ما میزان پراکندگی داده‌های آماری جمع‌آوری شده را نشان می‌دهد. به بیان دیگر، واریانس اطلاعاتی را در مورد میزان تغییر مقدار داده‌های آماری بیان می‌کند. هرچه مقدار واریانس بزرگ‌تر باشد، میزان پراکندگی و تغییر داده‌های آماری نیز بیشتر خواهد بود. دو فرمول واریانس:

$$Variance = \sum \frac{(x_i - \hat{x})^2}{n-1} \qquad Variance = \sum \frac{(x_i - \hat{x})^2}{n}$$

x_i: $the\ value\ of\ the\ one\ observation$

$\hat{x}$: $the\ mean\ value\ of\ all\ observations$

n: $the\ number\ of\ observations$

Sample: 1,2,3,4,5 avrage = 3

$$v = \frac{(1-3)^2 + (2-3)^2 + (3-3)^2 + (4-3)^2 + (5-3)^2}{5-1} = 2.5$$

$$v = \frac{(1-3)^2 + (2-3)^2 + (3-3)^2 + (4-3)^2 + (5-3)^2}{5} = 2$$

واریانس میانگین تفاوت هر نقطه با میانگین داده‌ها را نشان می دهد.

انحراف از معیار:

انحراف معیار به ما نشان می‌دهد که چگونه داده‌های آماری جمع‌آوری شده حول میانگین پراکنده شده‌اند. **اگر انحراف از معیار یک سری داده برابر با 5 شود، به این معنی است که بیشتر داده ها در فاصله 5 =+ واحدی از میانگین آنها قرار دارند**. فرمول:

$$std(standard\ Deviation) = \sqrt{Variance}$$

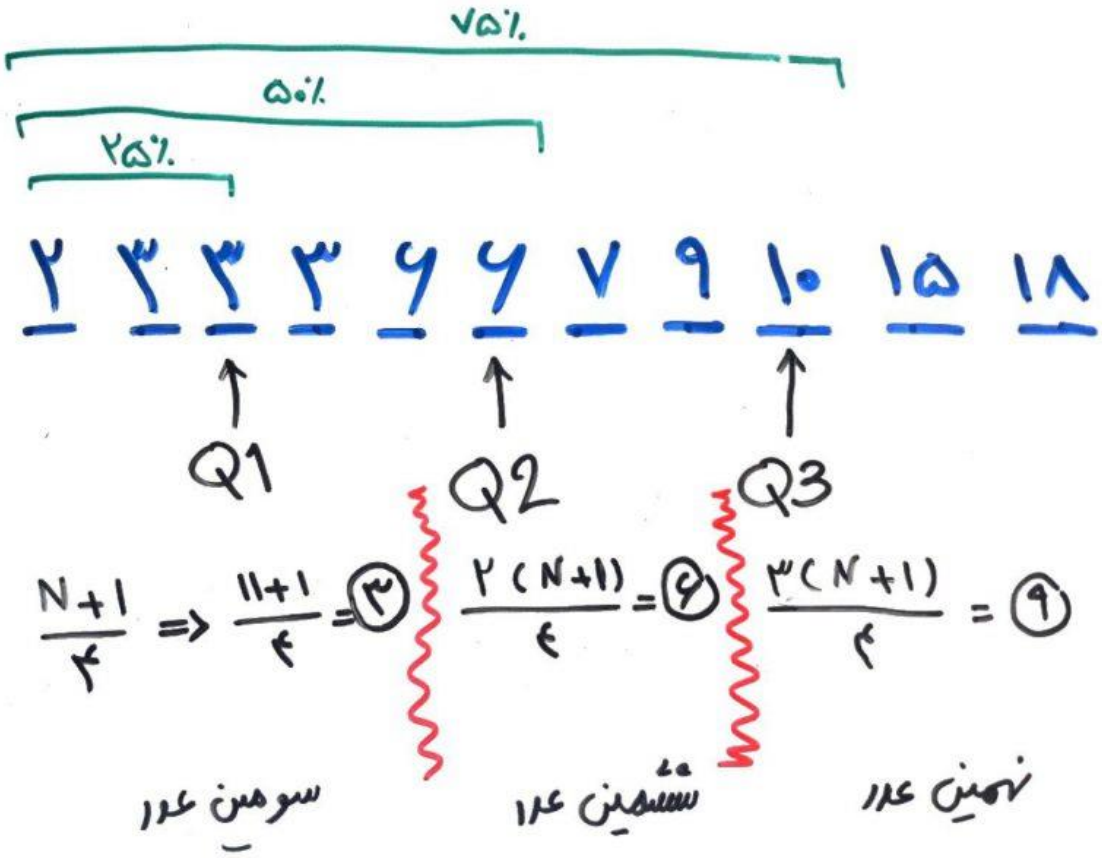
انحراف از معیار داده های مثال قبل با واریانس 2 برابر است با 1.41

چارک (Quartile) و IQR

چارک یا همان quartile داده‌ها را به چهار قسمت تقسیم می‌کند به گونه‌ای که هر قسمت ۲۵درصد (یک چهارم) داده‌ها را در خود داشته باشد. فرض کنید داده‌های زیر، امتیازاتی باشد که کاربران مختلف به یک راننده در تاکسی اینترنتی (بین صفر تا بیست) داده‌اند:

۶ – ۳ – ۲ – ۶ – ۳ – ۷ – ۱۵ – ۳ – ۹ – ۱۰ – ۱۸

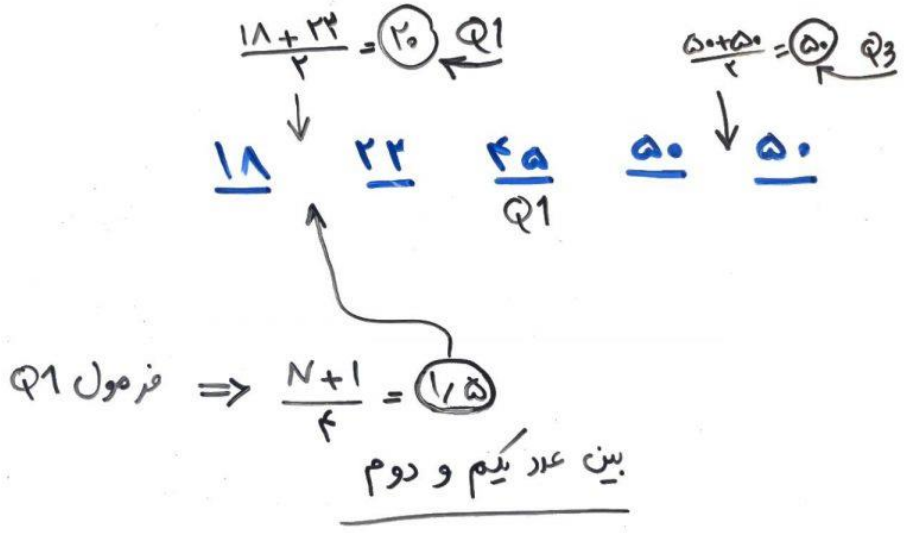
برای اینکه این داده‌ها را چارک‌بندی کنیم، ابتدا آن‌ها را به ترتیب از کوچک به بزرگ مرتب کرده و داده‌ها به به بخش‌های ۲۵ درصدی تقسیم می‌کنیم. چیزی مانند شکل زیر:



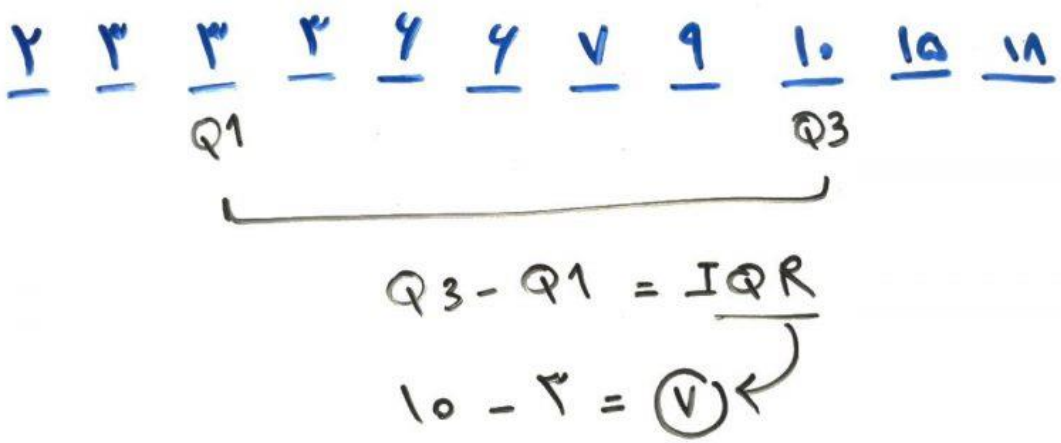
فرمول ساده است. داده‌ها را به علاوه‌ی یک کرده و سپس تقسیم بر چهار می‌کنیم تا به چارک اول برسیم. مثلاً در این مثال ۱۱ + ۱ برابر ۱۲ می‌شود و سپس ۱۲ را تقسیم بر ۴ می‌کنیم که به عدد ۳ می‌رسیم. پس باید داده‌ها سه تا سه تا تقسیم شود.

در کل سه چارک داریم، Q1، Q2 و Q3. چارک اول یا همان Q1، عددی است که ۲۵ درصد داده‌ها کمتر از آن هستند و ۷۵ درصد داده‌ها بیشتر از آن هستند. Q2 که چارک دوم است، میانه‌ی (median) داده‌ها را نشان می‌دهد. میانه که در دروس قبلی به آن اشاره کردیم همان عددی است که ۵۰ درصد داده‌ها کمتر از آن هستند و ۵۰ درصد داده‌ها بیشتر از آن. Q3 هم که همان‌طور که حدس می‌زنید، داده‌ای است که ۷۵ درصد داده‌ها کمتر از آن و ۲۵ درصد داده‌ها بیشتر از آن هستند.

البته نکته اینجاست که اگر داده‌ها به صورتی باشند که با استفاده از فرمولِ (N+1)/4 به عدد صحیح نرسند، بایستی میانگین آن دو عددی که به آن می‌رسیم را به عنوان چارک‌ها انتخاب کنیم. برای مثال شکل زیر را نگاه کنید که در آن چارک اول عددی بین ۱۸ و ۲۲ شده است که بایستی میانگین ۱۸ و ۲۲ را محاسبه کرده و آن را به عنوان چارک اول (Q1) در نظر گرفت:

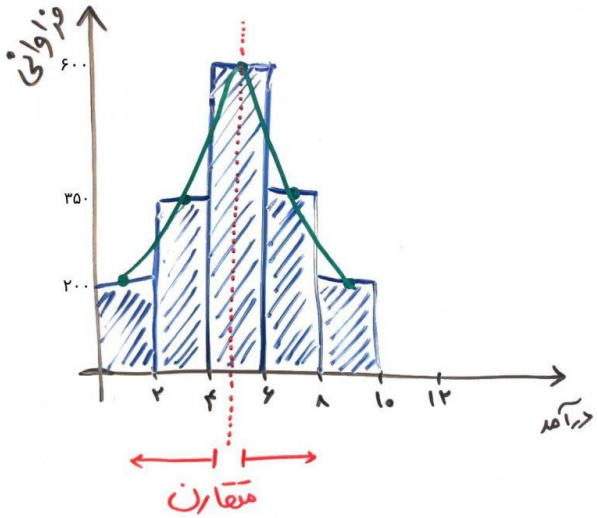


در اینجا تعریفی دیگر هم داریم. IQR (Interquartile Range) یا همان فاصله‌ی بین چارکی، به فاصله‌ی بین چارک اول و سوم می‌گویند. IQR به نوعی پراکندگی داده‌ها را بدون نویز (داده‌های پرت) نشان می‌دهد. برای مثال در شکل اول همین درس، IQR برابر ۷ می‌شود. مانند شکل زیر:

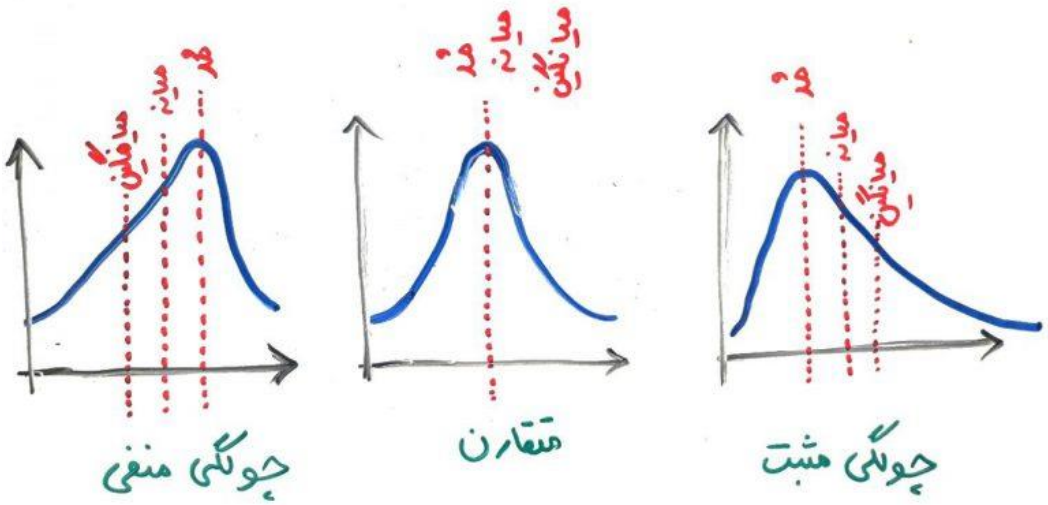


چولگی (Skewness)

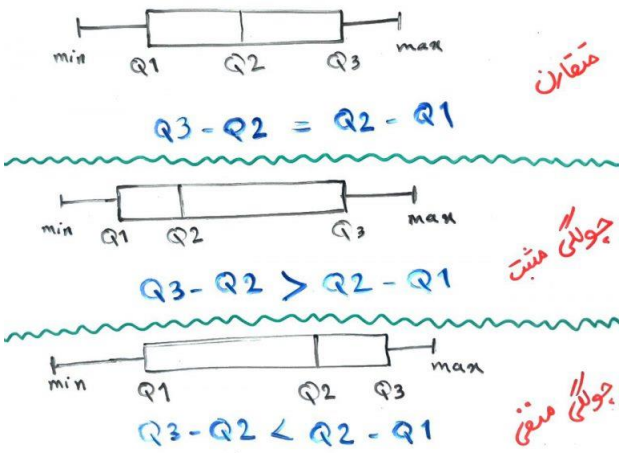
اگر توزیع داده‌ها به سمت راست یا چپ کشیده شده باشد، می‌گوییم داده‌ها چولگی یا همان skewness دارند. فرض کنید توزیعِ درآمد ماهیانه برای راننده‌های مختلف یک تاکسی اینترنتی به صورت شکل زیر باشد:



همان‌طور که مشاهده می‌کنید، توزیع داده به سمت راست کشیده شده است. تفسیر شکل این صورت است که تعدادی از راننده‌ها بوده‌اند که درآمدهای بالایی نسبت به عرف راننده‌ها (که همان بین ۴ تا ۶ میلیون است) داشته‌اند. اگر داده‌ها به این صورت پراکنده شده باشند، می‌گوییم توزیع داده به سمت راست (مثبت) چولگی دارد.

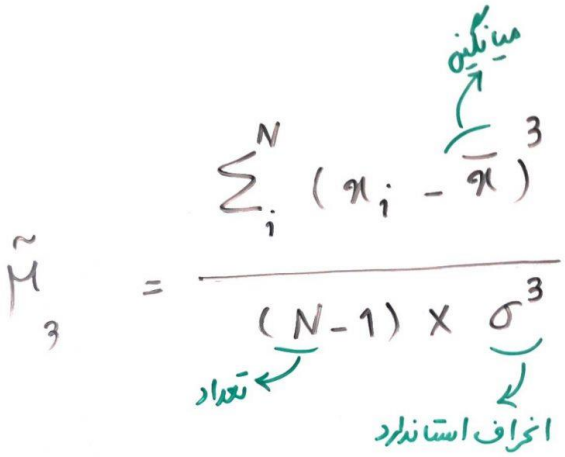


با استفاده از چارک‌ها (quartiles) نیز می‌توان چولگی داده‌ها را تشخیص داد. شکل زیر را در نظر بگیرید:



در قسمت اول از شکل بالا، داده‌ها نرمال هستند، چون چارک‌ها در حالت متوازن قرار دارند، ولی در قسمت دوم، مشخص می‌شود که داده‌ها به سمت راست چولگی دارد. در واقع اگر اختلاف Q3 و Q2 از اختلاف Q2 و Q1 بیشتر باشد، به این نتیجه می‌رسیم که داده‌ها به سمت راست (مثبت) چولگی دارند و اگر مانند قسمت سوم، برعکسِ این اتفاق بیوفتد به این نتیجه می‌رسیم که داده ها به سمت چپ (منفی) چولگی دارند.

البته اگر بخواهیم به صورت دقیق‌تر چولگی داده‌ها را بررسی کنیم، می‌توانیم از فرمول زیر استفاده کنیم:



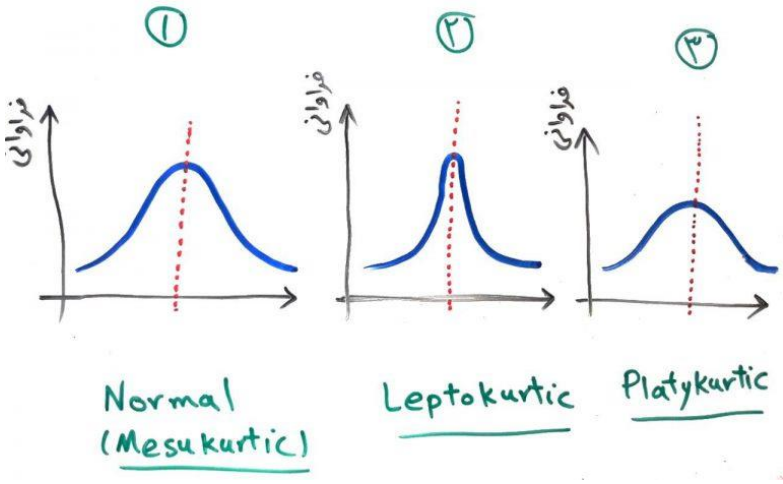
در این فرمول هر کدام از اعداد را منهای میانگین کرده، به توان ۳ می‌رسانیم و همه‌ی نتایج را با هم جمع می‌کنیم. سپس این داده‌ها را تقسیم بر تعداد کل داده‌ها منهای یک ضرب در انحراف استاندارد داده‌ها می‌کنیم. عددی که به دست می‌آید و اگر مثبت باشد، یعنی چولگی به سمت راست داریم و اگر منفی باشد یعنی چولگی به سمت چپ داریم. اگر صفر باشد به این این معنی است که داده‌ها چولگی ندارند.

چولگی در داده‌ها (skewness) را به عنوان گشتاور سوم نیز می‌شناسند. وجود توان ۳ در فرمول همین موضوع را می‌رساند. همان‌طور که در فرمول واریانس توان ۲ وجود داشت و واریانس را به عنوان گشتاور دوم می‌شناسند. همچنین گشتاور اول همان میانگین است که توان ۱ در فرمول خود دارد.

کشیدگی یا برجستگی (kurtosis)

کشیدگی یا برجستگی (kurtosis) گشتاور چهارم در یک مجموعه‌ی داده است که میزان برجستگی قله را در توزیع یک مجموعه‌ی داده مشخص می‌کند.

فرض کنید سه مجموعه‌ی داده داریم با توزیع‌های زیر:



شکل شماره‌ی ۱ (سمت چپ) توزیعِ یک مجموعه‌ی داده‌ی نرمال است. یعنی قله‌ی آن (همان مُد – mode) خیلی کم یا خیلی زیاد نیست و به سمت راست یا چپ هم چولگی ندارد. اما شکل شماره‌ی ۲ (وسط) قله یا همان مُد (mode) تیزتر از حد معمول است. به این حالت که قله تیزتر از حد معمول باشد kurtosis بالا می‌گویند. در حالی که شکل شماره‌ی ۳ (سمت راست) به دلیل اینکه قله‌ی کوتاهی دارد، kurtosis پایینی هم دارد. پس kurtosis به معنای تیزی، یا برجستگی قله (مُد) است و هر چقدر این مقدار بیشتر باشد، تیزیِ قله نیز بیشتر خواهد شد.

در آمار معمولاً به جای برجستگی (kurtosis) از برجستگی مازاد (excess kurtosis) استفاده می‌کنند. در واقع برجستگی برای داده‌های کاملاً نرمال برابر ۳ است. ولی در برجستگی مازاد یا همان excess kurtosis ما یک منهای ۳ (۳-) به فرمول اضافه می‌کنیم تا برجستگیِ داده‌های نرمال (مانند شکل بالا سمت چپ) برابر با صفر شود. پس داده‌هایی که از حالت نرمال قله‌ی تیزتری دارند، دارای برجستگیِ مازاد (excess kurtosis) مثبت هستند و داده‌هایی که از حالت نرمال پهن‌ترند، برجستگیِ مازادِ منفی خواهند داشت. فرمول برجستگیِ مازاد به صورت زیر است:

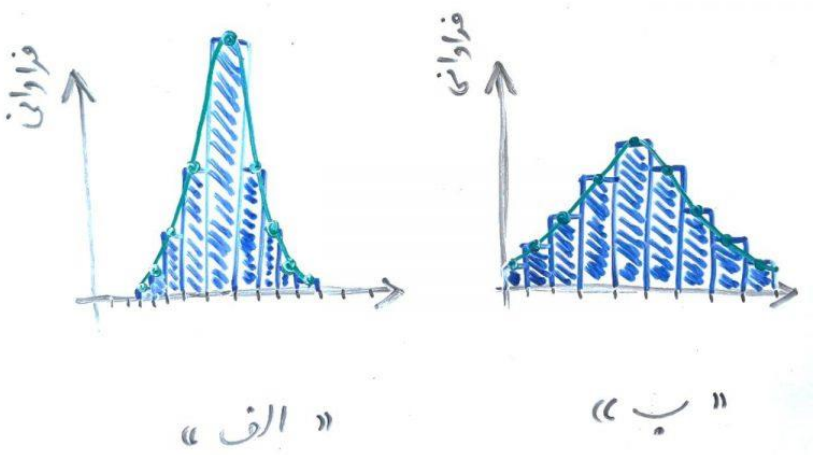
$$\text{excess kurtosis} = n \times \frac{\sum_i^n (x_i - \bar{x}_i)^4}{\sum_i^n (x_i - \bar{x}_i^2)^2} - 3$$

در کل با توجه به فرمول بالا، سه نوع برجستگی (kurtosis) در داده‌ها موجود است:

1. برجستگی Mesukurtic:
   a. اگر برجستگیِ مازاد یا همان excess kurtosis برابر صفر باشد یعنی قله در توزیع داده‌ها به صورت نرمال باشد، حالت Mesukurtic داریم
2. برجستگی Leptokurtic:
   a. اگر برجستگیِ مازاد یا همان excess kurtosis بالاتر از صفر باشد، یعنی قله در توزیع داده‌ها تیزتر از حالت نرمال باشد، برجستگی Leptokurtic رخ داده است
3. برجستگی Platykurtic:
   a. اگر برجستگیِ مازاد یا همان excess kurtosis کمتر از صفر باشد، یعنی قله در توزیع داده‌ها پهن‌تر از حالت نرمال باشد، برجستگی Platykurtic رخ داده است

به برجستگی (kurtosis) گشتاور چهارم نیز گفته می‌شود چون در فرمولش توان چهار موجودست.

برجستگی زیاد (leptokurtic) نشان می‌دهد که داده‌های ما به صورت متراکم نزدیک به قله جمع شده‌اند. برای مثال فرض کنید شکل زیر، توزیع سود دو شرکت در سالیان مختلف است:

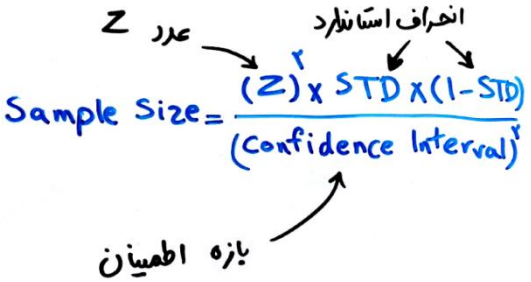


همان‌طور که می‌بینید توزیع سودِ شرکت «الف»، به نوعی leptokurtic محسوب می‌شود در حالی که توزیع سودِ شرکت «ب»، playkurtic است. سرمایه‌گذاران معمولاً با مشاهده‌ی شرکت «ب» درمیابند که ریسک سرمایه‌گذاری در آن بالاست زیرا قابلیت پیش‌بینی کمتری دارد. در واقع احتمال اینکه سود خیلی زیاد یا سود خیلی کم از آن بگیرند، بیشتر است. ولی سرمایه‌گذاری در شرکت «الف» ریسک کمتری دارد چون پیش‌بینی‌پذیر تر است. البته برای مثال یک سرمایه‌گذار می‌تواند بخشی از پولش را در شرکت‌هایی سرمایه‌گذاری کند که سود آن‌ها leptokurtic هستند و بخشی دیگر را برای اطمینان در شرکت‌هایی با توزیع سود platykurtic سرمایه‌گذاری کند.

نمونه‌گیری آماری و محاسبه‌ی حداقل تعداد نمونه (Min Sample Size)

فرض کنید می‌خواهیم نظر مردم شهر اهواز را در مورد شهردار این شهر بدانیم. یک پرسشنامه طراحی می‌کنیم و آن را به تعدادی از شهروندان اهوازی داده تا به آن پاسخ دهند. اولین سوالی که احتمالاً ذهن ما را درگیر خود می‌کند، این است که از چه تعداد از شهروندان بخواهیم پرسشنامه را پاسخ دهند؟ پاسخ واضح است، هر چه بیشتر، بهتر. اما هر چقدر تعداد افرادی بیشتری در پاسخ به پرسشنامه درگیر باشند، زمان و هزینه‌ی بیشتری نیز بایستی صرف نظرخواهی از شهروندان شود. پس به دنبال راهی هستیم که حداقل تعداد نمونه‌ی مناسب که نظر آن‌ها بیان‌گر نظر کل مردم شهر باشد را پیدا کنیم. به این کار محاسبه‌ی حداقل تعداد نمونه (minimum sample size) می‌گویند که کاربردهای متعددی در پردازش داده‌ها دارد.

پیدا کردن نمونه‌ی مناسب با یک فرمول نسبتاً ساده آماری به دست می‌آید (فرمول‌های متعددی برای نمونه‌گیری موجود است که ما به یکی از ساده‌ترین آن‌ها اشاره می‌کنیم):



بازه‌ی اطمینان (Confidence Interval) یا حاشیه‌ی خطا (Margin of Error)

همان‌طور که از اسم آن مشخص است، بازه‌ی اطمینان، مقدار عدم اطمینان یک نمونه را نسبت به جمعیت کل مشخص می‌کند. به زبان ساده یعنی چقدر به این نمونه شَک داریم. به بیان دیگر، مقدار بازه‌ی اطمینان یا حاشیه‌ی خطا به ما می‌گوید که چقدر اطمینان داریم نمونه‌ی گرفته شده، به کل جمعیت قابل تعمیم باشد.

برای مثال فرض کنید در همان مثال بالا (پرسشنامه‌ی میزان رضایت از شهردار اهواز)، بازه‌ی اطمینان را بر روی ۵ درصد تنظیم کنیم و نظر شهروندان (که به صورت نمونه انتخاب شده‌اند) هم ۶۰ درصد رضایت از شهردار باشد. به این ترتیب می‌توانیم بگوییم که نظر کل مردم شهر (جمعیت کل)، بین بازه‌ی ۵۷ تا ۶۳ (پنج‌درصد کمتر از شصت تا پنج‌درصد بیشتر از شصت) درصد رضایت است. در واقع برای تعمیم از نمونه به جمیعت، ۵ درصد خطا داریم.

سطح اطمینان

سطح اطمینان میزان احتمال درستی در تکرارهای زیاد از آزمایش است. برای مثال اگر سطح اطمینان را برابر ۹۵ درصد بگذاریم، در همان مثال قبلی (پرسشنامه‌ی میزان رضایت از شهردار اهواز)، به این معناست که در صد بار تکرار این آزمایش (صد بار که از گروه‌های مختلف شهروندان، نمونه گرفته و پرسش کنیم)، ۹۵ مرتبه، همان عددی به دست خواهد آمد که از جمعیت کل خواهیم گرفت.

چقدر این عدد بالاتر باشد، طبیعتاً تعداد نمونه‌های لازم نیست بیشتر است ولی معمولاً این عدد را ۹۵ یا ۹۹ می‌گذارند.

در فرمول بالا، سطح اطمینان وجود ندارد. در واقع بایستی با استفاده از جدول Z (که یک جدول معروف آماریست)، سطح اطمینان را به عدد Z تبدیل کنیم. جدول Z در کتاب‌های آماری وجود دارد و فعلاً نمی‌خواهیم به آن بپردازیم ولی برای همین درس، یک قسمت کوچک تبدیل سطح اطمینان به Z را در جدول زیر مشاهده می‌کنید:

سطح اطمینان	مقدار Z
۸۰٪	۱.۲۸
۹۰٪	۱.۶۵
۹۵٪	۱.۹۶
۹۹٪	۲.۵۸

برای مثال اگر بخواهیم سطح اطمینان را ۹۵ قرار دهیم، در فرمول بالا به جای Z، عدد ۱.۹۶ قرار می‌گیرد.

Numpy

Numpy یک کتابخانه قابل افزودن به پایتون است که کاربرد اصلی‌اش در مقاصد علمی و برای کار با اعداد است. پایتون به صورت پیش‌فرض تنها از آرایه‌ها و متغیرها برای عملیات ریاضی ساده پشتیبانی می‌کند .بستهٔ نام‌پای برای کار با اعداد از راه ماتریس‌ها و آرایه‌های چندبعدی طراحی شده است.

فراخوانی این کتابخانه

```
import numpy as np
```

نامپای تابع های زیادی برای اعمال ریاضی دارد که برای مطلع شدن از این توابع می توان از کد زیر استفاده کرد

```
dir(np)
```

برخی توابع مهم ناپای

تبدیل لیست به آرایه نامپای

```
np.array([1,2,3])
```

ساخت آرایه و ماتریس در ابعاد بالا با مقدار صفر

Np.zeros(shape, dtype)

```
np.zeros(7)
np.zeros((5,7))
np.zeros((2,5,7))
np.zeros((3,2,5,7), dtype="int")
```

ساخت آرایه و ماتریس در ابعاد بالا با مقدار یک

Np.ones(shape, dtype)

```
np.ones(7)
np.ones((5,7))
np.ones((2,5,7))
np.ones((3,2,5,7), dtype="int")
```

ساخت آرایه با مقدار 6

```
np.ones(7) * 6
```

ساخته آرایه با ابعاد مختلف با یک مقدار دلخواه

Np.full(shape , number, dtype)

```
np.full(7,3)
np.full((5,7), 3)
np.full((2,5,7), 3)
np.full((3,2,5,7), 3, dtype="float")
```

ساخت یک آرایه با لیستی از اعداد ترتیبی (مرتب)

Np.arange(start , stop , step)

```
np.arange(0,10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

ساخت یک آرایه از 0 تا 100 به صورتی که تعداد اعضای آرایه 6 باشد

Np.linspace(start , stop , number)

```
np.linspace(0,100,6)
```

```
array([  0.,  20.,  40.,  60.,  80., 100.])
```

اعداد تصادفی (Random)

ساخت آرایه با اعداد تصادفی

Np.random.rand(shape)

```
np.random.rand(5)
```

```
array([0.41923568, 0.07129158, 0.31673463, 0.27463542, 0.08996385])
```

```
np.random.rand(5,7)
np.random.rand(5,7,8)
```

ساخت آرایه با اعداد تصادفی صحیح (Z)

Np.random.randn(shape)

```
np.random.randn(5)
```

```
array([-0.29635448, -0.78964874, -0.43079313,  0.2206741 ,  1.7669638 ])
```

ساخت آرایه با اعداد تصادفی طبیعی (N)

Np.random.randint(min, max, shape)

```
np.random.randint(1,100,(5,2))
```

توزیع نرمال

توزیع نُرمال (به انگلیسی: Normal distribution)، توزیع گاوسی (به انگلیسی: Gaussian) یا توزیع به‌هنجار ، یکی از مهم‌ترین توزیع‌های احتمال پیوسته در نظریه احتمالات است. دلیل این نام‌گذاری و نیز اهمیت این توزیع، این است که اُفت‌وخیز بسیاری از کمیت‌های طبیعی (فیزیکی) حول یک مقدار ثابت، از این توزیع پیروی می‌کند.

Np.random.normal(location , std, shape)

```
np.random.normal(20, 3, (4,3))
```

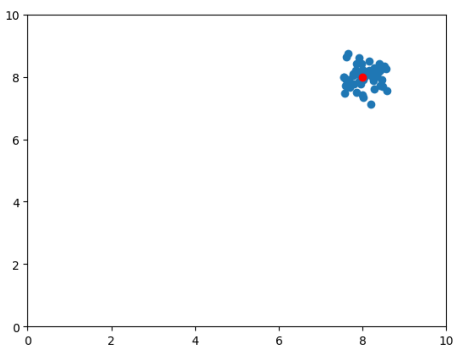
فرض کنید می خواهیم یک 50 عدد در اطراف عدد 8 به صورت تصادفی بنویسیم، داریم

```
np.random.normal(8, 0.1, 50)
```

نکته: هرچه مقدار std(standard deviation) کمتر باشد اعداد تصادفی ساخته شده، به عدد location ما نزدیک تراند.

اگر بخوایم تعدادی نقطه تصادفی در نمودار مختصات در نزدیکی نقطه (8,8) بکشیم میتوانیم از کد زیر استفاده کنیم.

```
x = np.random.normal(8, 0.3, 50)
y = np.random.normal(8, 0.3, 50)
```



دریافت اطلاعات یک آرایه

ابتدا یک ماتریس می‌سازیم

```
matrix = np.random.randint(10,50, (5,4))
```

نمایش تعداد بعد های ماتریس

```
matrix.ndim
```

نمایش شکل ماتریس

```
matrix.shape
```

نمایش تعداد اعداد ماتریس

```
matrix.size
```

نمایش نوع اعداد ماتریس

```
matrix.dtype
```

تابع reshape: تغییر حالت یک آرایه

numpyArray.reshape((shape))

```
example = np.arange(1,16)
example
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
example.reshape((3,5))
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

تابع max: مشخص کردن بزرگ ترین عدد یک آرایه

```
matrix.max()
```

تابع argmax: پیدا کردن شماره بزرگترین عدد آرایه

```
matrix.argmax()
```

به همین صورت min و argmin را هم برای کوچک ترین عدد ارایه داریم.

تابع concatenate: این تابع میتواند تعداد زیادی آرایه را که ابعاد برابری دارند به هم وصل کند

```
first_array  = np.random.randint(1,7, (3,2))
second_array = np.random.randint(1,7, (3,2))

third_array  = np.concatenate([first_array, second_array])
fourth_array = np.concatenate([first_array, second_array], axis = 1)
```

وقتی از axis = 1 استفاده کریم یعنی دوتا آرایه به صورت ستونی به هم وصل شوند.

مرتب یا sort کردن آرایه

```
array = np.random.randint(1,20,7)
array
```

```
array([10, 19, 12,  5,  3,  4,  3])
```

```
new_array = np.sort(array)
new_array
```

```
array([ 3,  3,  4,  5, 10, 12, 19])
```

اگر آرایه چند بعدی داشته باشیم با آرگومان axis میتوانیم مشخص کنیم که ستونی مرتب شود یا سطری، axis = 1 برای ستونی و axis = 0 برای سطری.

عملیات ها و شرط ها در numpy

فرض کنید آرایه زیر را داریم

```
array = np.random.randint(1,100,12)
array
```

```
array([24, 21,  7, 76, 29, 82, 68, 91, 38, 81, 74, 13])
```

حالا میخوام اونایی رو انتخاب کنیم که از 50 بزرگتر باشند.

```
array[array > 50]
```

```
array([76, 82, 68, 91, 81, 74])
```

میتونیم به صورت زیر هم بنویسیمش

```
condition = array <= 40
array[condition]
```

```
array([24, 21,  7, 29, 38, 13])
```

میتونیم عملیات های زیر رو روی ماتریس یا ارایمون انجام بدیم

```
array * 2
array - 2
array + 2
array / 2
array * array
```

میانگین آرایه

```
np.mean(array)
```

بدست آوردن میانه آرایه: میانه عددی است که در وسط آرایه قرار دارد (ابتدا آرایه از کوچک به بزرگ مرتب میشود)

```
np.median(array)
```

انحراف از معیار یا std (standard deviation)

```
np.std(array)
```

Pandas

Pandas یک کتابخانه برای دستکاری و با داده هاست، با Pandas میتوانیم دیتافریم های مختلفی را به صورت فایل های مختلف باز کنیم و تغییرات مورد نظرمان را اعمال کنیم.

سری ها در pandas

```
pd.Series([10,20,30,40,50,60])
```

```
0  10
1  20
2  30
3  40
4  50
5  60
dtype: int64
```

ساخت دیتافریم در Pandas

```
list_example = [5,10,56,98,45]
pd.DataFrame(columns=['Values'], data = list_example)
```

	Values
0	5
1	10
2	56
3	98
4	45

```
vals = np.arange(1,13).reshape(3,4)
pd.DataFrame(data = vals, columns = ['a','b','c','d'])
```

	a	b	c	d
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

خواندن یک دیتاست به صورت فایل csv

```
df = pd.read_csv("data file name.csv")
```

خواندن یک دیتاست به صورت فایل excel

```
df = pd.read_excel("data file name.csv")
```

نمایش چند سطر اول

```
df.head()
# df.head(5)
# df.head(10)
```

نمایش شکل دیتافریم

```
df.shape
```

نمایش ابعاد دیتافریم

```
df.ndim
```

نمایش مقادیر یک ستون از دیتافریم

روش اول

```
df['column_1']
```

روش دوم

```
df.column_1
```

اگر فقط دیتای اون ستون رو بخوایم میتونیم از کد زیر استفاده کنیم

```
df['column_1'].values
```

نمونه هایی از نمایش به صورت slicing

Val1 ستون است و B و D سطر ها و 1 و 4 شماره سطر ها هستند.

```
df[["VAL1"]]["B":"D"]
```

	VAL1
B	-0.319318
C	0.528813
D	0.955057

```
df[["VAL1"]][1:4]
```

	VAL1
B	-0.319318
C	0.528813
D	0.955057

انتخاب دوتا از ستون ها

```
df[['column_1','column_2']]
```

انتخاب ستون اول تا پنجم

```
df['column_1':'column_5']
```

استفاده از loc و iloc برای indexing و slicing

iloc به معنی integer location است، پس در استفاده از آن به جای اسم سطر و ستون از شماره آن ها استفاده می کنیم.

انتخاب اولین سطر دیتافریم با loc و iloc

```
# select first row using loc
print(df.loc['index_1_name'])

# select first row using iloc
print(df.iloc[0])
```

انتخاب (slice) کردن از سطر اول تا دهم

```
# select first row to tenth row using loc
print(df.loc['index_1_name':'index_10_name'])

# select first row to tenth row using iloc
print(df.iloc[0:10])
```

انتخاب تمام سطرهای ستون یک تا پنج

در آرگومان اول : گذاشتیم یعنی تمام سطر ها و در آرگومان بعدی مشخص کردیم از ستون یک تا 5

```
df_slice = df.iloc[:, 1:6]
```

```
df_slice = df.loc[:, 'column_1_name': 'column_5_name']
```

عملیات ها روی دیتافریم

اضافه کردن ستون جدید

فرض کنید از ستون یک تا چهار را داریم که هر کدام پنج سطر دارند

```
# add a new column
df['col_5'] = [65, 8, 94, 23, 87]
```

```
df ['col_6'] = df['col_1'] + df['col_5']
```

حذف یک ستون یا سطر

با axis مشخص می کنیم ستون حذف شود یا سطر، axis = 1 برای ستون و 0 برای سطر.

df.drop([names], axis = 1 or 0, inplace = true or false)

```
df.drop(['col_1','col_2','col_3'], axis = 1)
```

حذف سطر 1 تا 5132

```
new_df = df.drop(index=range(1, 5133))
```

حذف چند سطر انتخابی

```
new_df = df.drop([1, 65, 88])
```

اگر بخواهیم همون دیتافریم اولیمون ازش یه چیزایی حذف بشه و توی دیتافریم جدید نریزیم میتونیم از کد زیر استفاده کنیم.

```
df.drop([1, 65, 88], inplace=True)
```

یا

```
df = df.drop([1, 65, 88])
```

مقادیر null در DataFrame

مشخص کردن تعداد مقادیر null در دیتافریم

```
df.isnull().sum()
```

```
survived       0
pclass         0
sex            0
age          177
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
deck         688
embark_town    2
alive          0
alone          0
```

مشخص کردن درصد مقادیر null نسبت به طول دیتافریم

```
(df.isnull().sum() * 100) / len(df)
```

```
survived       0.000000
pclass         0.000000
sex            0.000000
age           19.865320
sibsp          0.000000
parch          0.000000
fare           0.000000
embarked       0.224467
class          0.000000
who            0.000000
adult_male     0.000000
deck          77.216611
embark_town    0.224467
```

```
alive       0.000000
alone       0.000000
dtype: float64
```

میخواهیم همه سطر ها و ستون های دیتا فریم را بگیریم با شرط اینکه ستون اول آنها null نباشد

```
df[df.iloc[:, 0].notnull()]
```

یا به صورت زیر

```
df[df.iloc[:, 0].notna()]
```

حذف سطر هایی که مقدار Nan دارند یا Null هستند

```
df.dropna(inplace=True)
```

حذف ستون هایی که مقدار Nan دارند یا Null هستند

```
df.dropna(axis=1, inplace=True)
```

پر کردن مقادری null با یک مقدار خاص

```
df.fillna(1000)
```

یا

```
df.fillna("test")
```

معمولا در ستون های عددی باید مقادیر nan یا null با میانگین اعداد موجود پر شوند، و یا در ستون های عددی باید با mode آن ستون پر شوند.

Mode: پر تکرار ترین کلمه یا کتگوری در آن ستون

```
# Identify categorical and numerical columns
categorical_cols = df.select_dtypes(exclude=np.number).columns
numerical_cols = df.select_dtypes(include=np.number).columns

# Fill null values in categorical columns with the mode
for col in categorical_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)

# Fill null values in numerical columns with the mean
for col in numerical_cols:
    df[col].fillna(df[col].mean(), inplace=True)
```

توابع Pandas

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    object 
 8   name          398 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

```
df.shape
```

```
(398, 9)
```

محاسبه میانگین ستون های عددی

```
num_cols = df.select_dtypes(include=np.number).columns
df[num_cols].mean()
```

محاسبه انحراف از معیار ستون های عددی

```
num_cols = df.select_dtypes(include=np.number).columns
df[num_cols].std()
```

محاسبه میانه(median)

```
num_cols = df.select_dtypes(include=np.number).columns
df[num_cols].median()
```

تعداد هر کدام از مقادیر یک کلا در یک ستون

```
df['cylinders'].value_counts()
```

```
cylinders

4   204
8   103
6    84
3     4
5     3
Name: count, dtype: int64
```

مشخص کردن لیست مقادیر یک ستون

```
df['cylinders'].unique()
```

```
array([8, 4, 6, 3, 5])
```

```
df['origin'].unique()
```

```
array(['usa', 'japan', 'europe'], dtype=object)
```

گروهبندی دو ستون

```
df.groupby('origin')['cylinders'].mean()
```

```
origin
europe   4.157143
japan    4.101266
usa      6.248996
Name: cylinders, dtype: float64
```

```
pd.crosstab(df['origin'], df['cylinders'])
```

cylinders	3	4	5	6	8
origin					
europe	0	63	3	4	0
japan	4	69	0	6	0
usa	0	72	0	74	103

گروهبندی با تمام ستون های عددی

```
num_cols = df.select_dtypes(include=np.number).columns
df.groupby('origin')[num_cols].mean()
```

origin	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
europe	27.891429	4.157143	109.142857	80.558824	2423.300000	16.787143	75.814286
japan	30.450633	4.101266	102.708861	79.835443	2221.227848	16.172152	77.443038
usa	20.083534	6.248996	245.901606	119.048980	3361.931727	15.033735	75.610442

تبدیل داده های متنی یک ستون به عددی به صورت دستی

```
origin = {'usa': 0, 'japan': 1, 'europe': 2}
df['origin'] = df['origin'].map(origin)
```

میتونیم عملیات بالا را به صورت اتوماتیک انجام بدیم

```
# Use factorize to transform the categorical column to numerical
df['origin'], class_mapping = pd.factorize(df['origin'])

print("Class mapping:", class_mapping.tolist())
df.head()
```

نرمالایز (Normalize) کردن داده ها به صورت دستی

```
def normalizer(x):
    return (x - x.min()) / (x.max() - x.min())

df.iloc[:, 0:8] = df.iloc[:, 0:8].transform(normalizer)
```

حالا داده های ستون 0 تا 7 ما با یک تابع احتمالی به اعداد بین 0 تا 1 تبدیل شدند.

توجه داشته باشید که قبل از انجام این کار برای همه ستون ها باید ستون های Categorical را به ستون های عددی تبدیل کنیم.

میتونیم داده هارو به صورت زیر هم نرمالایز کنیم (sine normalizer)

```
df.iloc[:, 0:8] = df.iloc[:, 0:8].transform(lambda a: np.sin(a))
```

Matplotlib

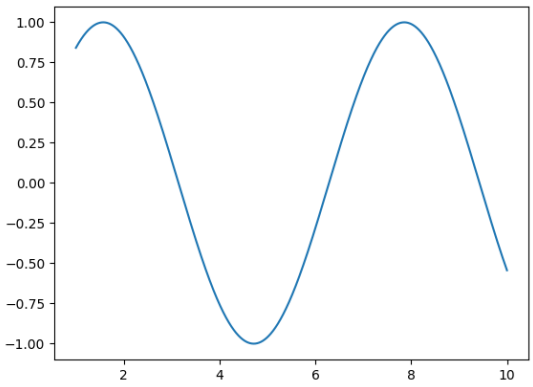
این کتابخانه ساخته شده برای به نمایش در آوردن و تجسم (visualization) داده ها.

```
import matplotlib.pyplot as plt
```

linear plot (رسم خطی)

plt.plot(x list, y list, color = “”)

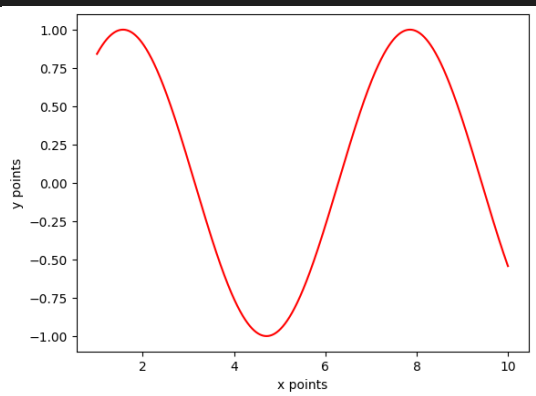
```
x_points = np.arange(1, 10, 0.03)
y_points = np.sin(x_points)

plt.plot(x_points, y_points)
```



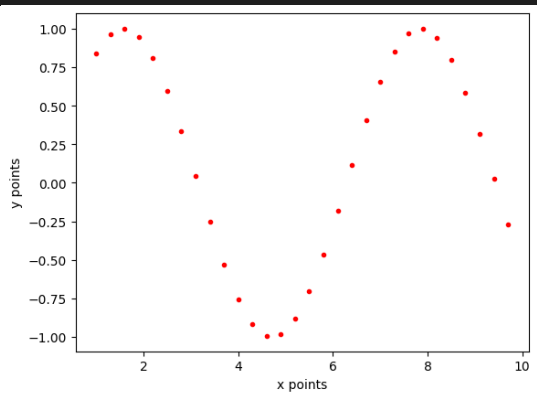
در کد بالا یک تعداد اعداد در رنج یک تا ده ساخته شده و سپس سینونس آنها را حساب کردیم و بعد با کتابخانه matplotlib آن را نمایش دادیم.

با کد زیر میتوانیم اسم محور ها را مشخص کنیم

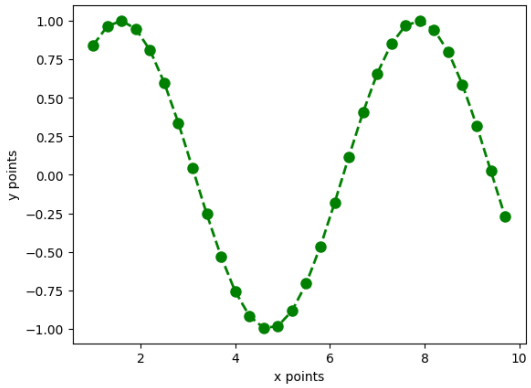
```
plt.xlabel('x points')
plt.ylabel('y points')

plt.plot(x_points, y_points,color = 'red')
```



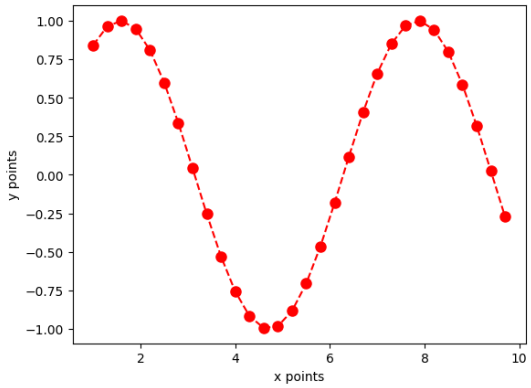
```
plt.plot(x_points, y_points,".", color = 'red')
```



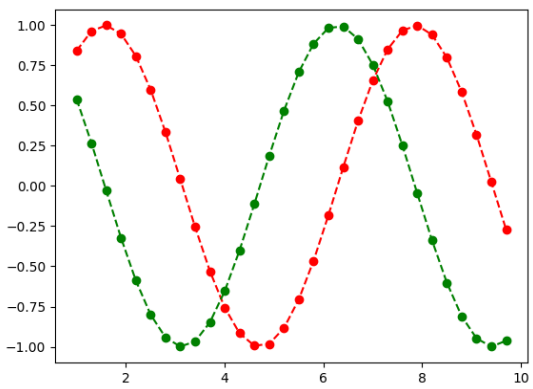

```
plt.plot(x_points, y_points, 'go--', linewidth=2, markersize=8)
```




```
plt.plot(x_points, y_points, 'ro--', linewidth=1.5, markersize=8)
```

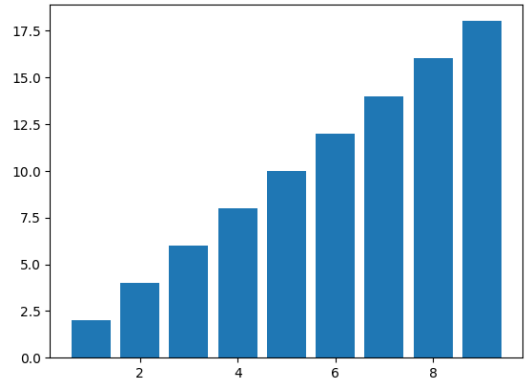


```
x_points = np.arange(1, 10, 0.3)
y_points_1 = np.sin(x_points)
y_points_2 = np.cos(x_points)

plt.plot(x_points, y_points_1, "ro--")
plt.plot(x_points, y_points_2, "go--")
```

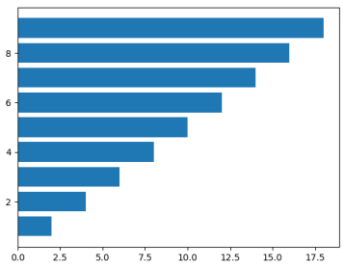


Bar plot (رسم ستونی)

```
x_points = np.arange(1, 10)
y_points = (x_points) * 2

plt.bar(x_points, y_points)
```

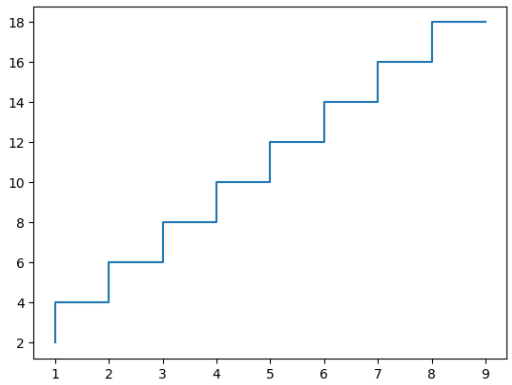



```
x_points = np.arange(1, 10)
y_points = (x_points) * 2

plt.barh(x_points, y_points)
```



Step Plot (رسم پله ای)

```
x_points = np.arange(1, 10)
y_points = (x_points) * 2

plt.step(x_points, y_points)
```

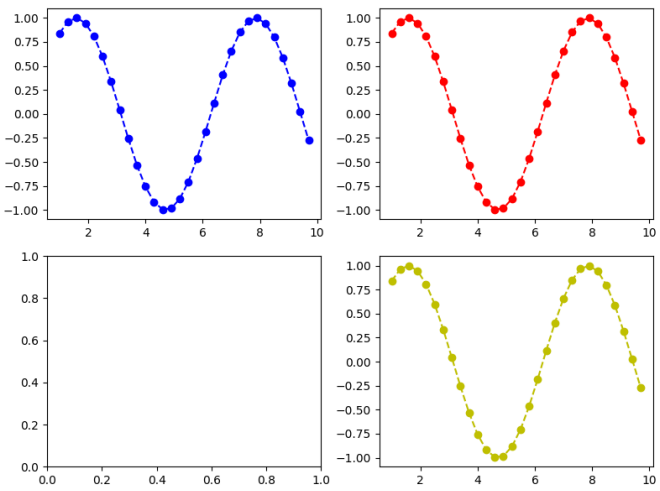


تغییر اندازه تصویر

```
plt.figure(figsize = (10,8))
plt.step(x_points, y_points)
```

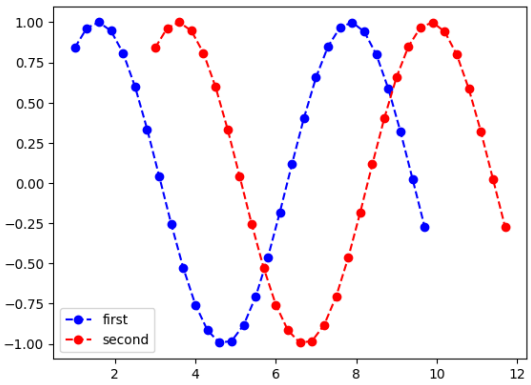
نمایش به صورت Grid بندی (نمایش چند نمودار) باهم

```
Figure, Axes = plt.subplots(nrows = 2, ncols = 2, figsize=(8, 6))
Axes[0, 0].plot(x_points, y_points, "bo--")
Axes[0, 1].plot(x_points, y_points, "ro--")
Axes[1, 1].plot(x_points, y_points, "yo--")

plt.tight_layout()
```



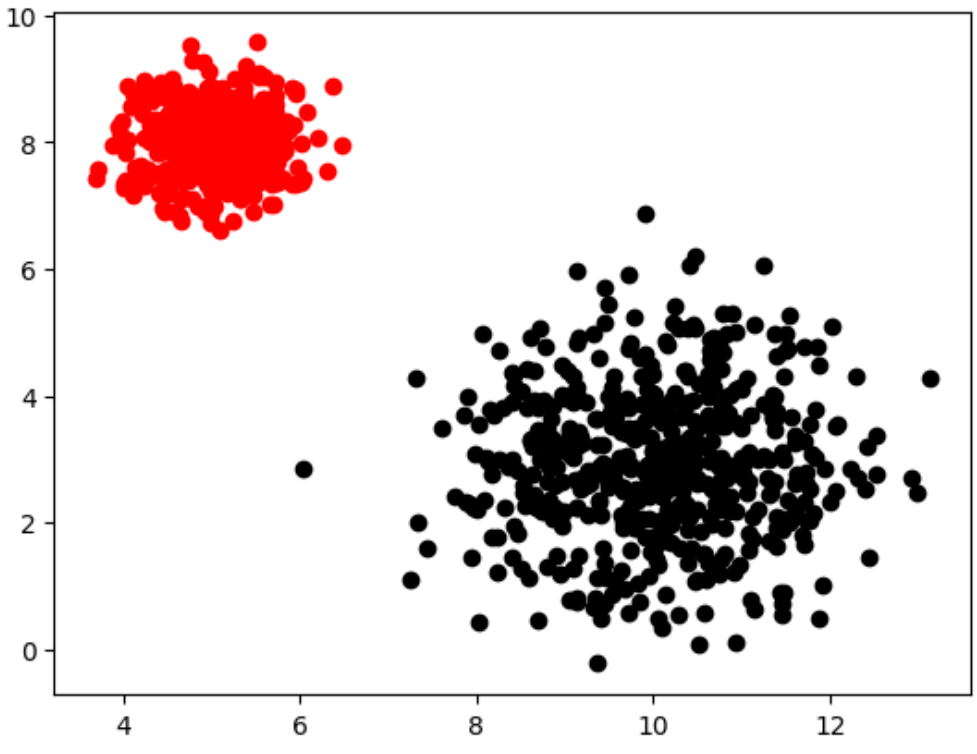
نمایش لیبل (label) در نمودار

```
plt.plot(x_points, y_points, "bo--", label="first")
plt.plot(x_points+2, y_points, "ro--", label="second")

plt.legend()
```



Scatter plot (رسم نقطه ای)

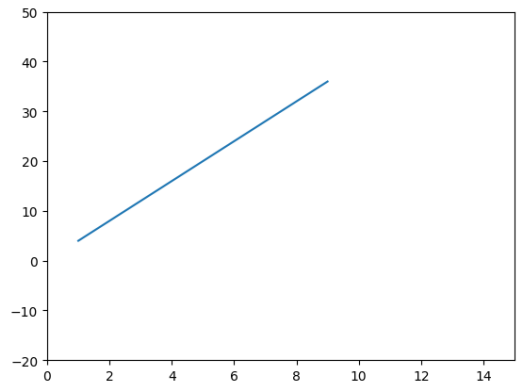
```
x_1 = np.random.normal(5, 0.5, 500)
y_1 = np.random.normal(8, 0.5, 500)

x_2 = np.random.normal(10, 1.2, 500)
y_2 = np.random.normal(3, 1.2, 500)


plt.scatter(x_1, y_1, color = "red")
plt.scatter(x_2, y_2, color = "black")
```



مشخص کردن بازه نمایش نمودار

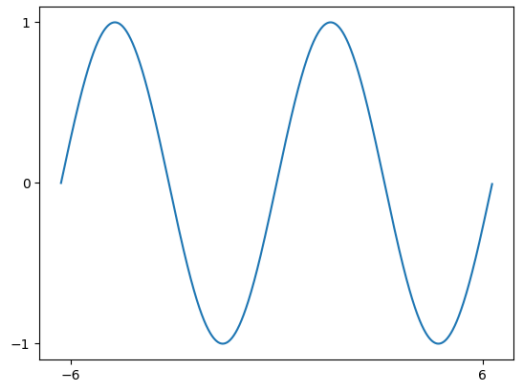
plt.axis([x min, x max, y min, y max])

```
x_points = np.arange(1,10)
y_points = x_points * 4

plt.axis([0,15,-20,50])
plt.plot(x_points, y_points)
```



مشخص کردن اعدادی که میخوایم نمایش داده شوند

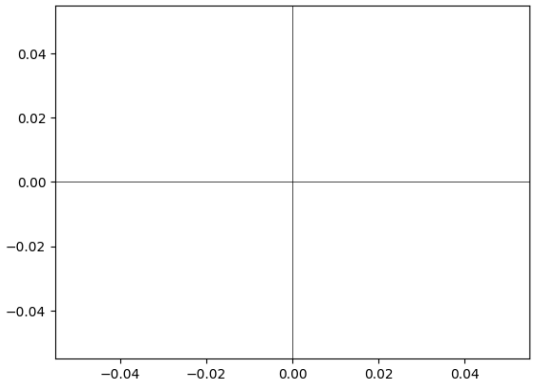
```
x = np.arange(-2 * np.pi, 2 * np.pi, 0.01)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)

ax.set_xticks([-6, 6])
ax.set_yticks([-1, 0, 1])

plt.show()
```



همانطور که مشخص است در نمودار x ها فقط اعداد 6- و 6 و در نمودار y اعداد 1- و 0 و 1 نمایش داده می شوند. میتوانیم در تابع set_xticks و set_yticks از string هم استفاده کنیم.

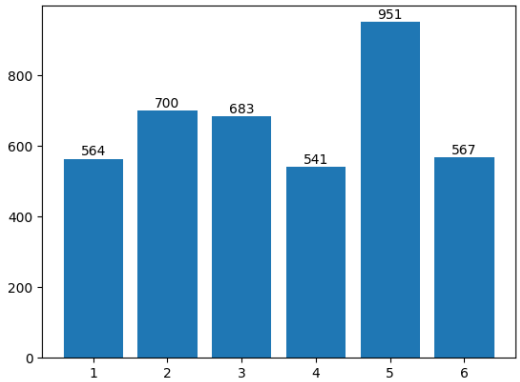
نمایش محور های مختصات

```
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.plot()
```



نمایش مقدار ارتفاع در bar plot

```
x = np.arange(1, 7)
y = np.random.randint(500, 1000, 6)

figure, ax = plt.subplots()
plot = ax.bar(x, y)

for bar in plot:
    height = bar.get_height()
    ax.text(x = bar.get_x() + bar.get_width() / 2, y = 1.002 * height, s =
int(height), ha = 'center', va = 'bottom')
```



Seaborn

این کتابخانه یک کتابخانه شبیه و کمکی برای **Matplotlib** است.

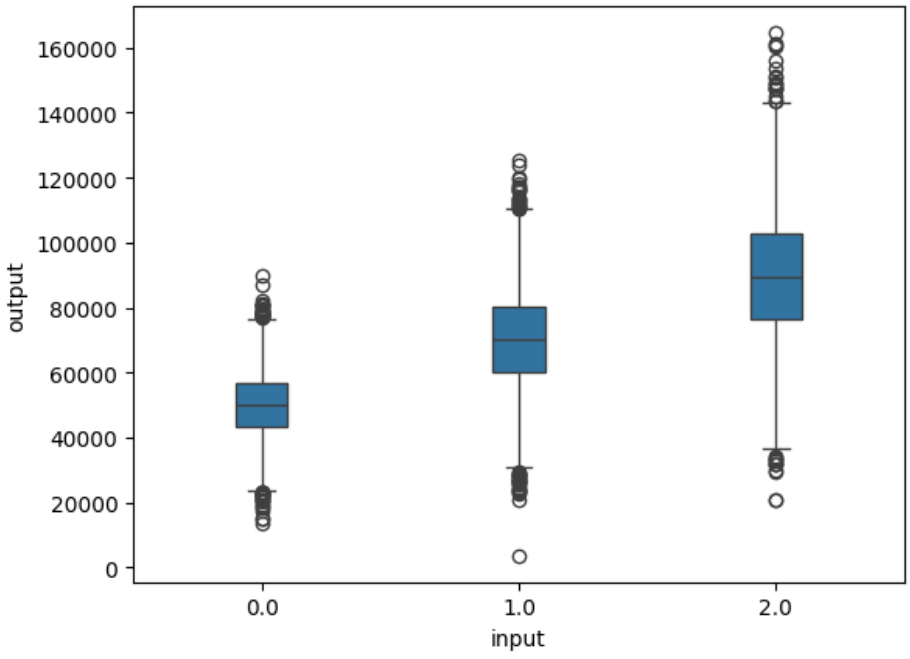
```
import seaborn as sns
```

فرض کنید یک دیتاست با دو ستون input و output داریم که ستون input دارای سه کلاس 0 و 1 و 2 است.

میخواهیم نمودار boxplot آن را نمایش دهیم.

نمودار جعبه‌ای یاBoxplot یک روش گرافیکی برای نمایش خلاصه‌ای از داده‌ها است. این نمودار پنج نقطه کلیدی از داده‌ها را نشان می‌دهد: کمینه(Minimum) ، چارک اول (Lower Quartile)، میانه (Median)، چارک سوم Upper Quartile و بیشینه(Maximum). این نقاط کلیدی به ترتیب 25 درصد، 50 درصد و 75 درصد داده‌ها را نشان می‌دهند. همچنین، نمودار جعبه‌ای می‌تواند نشان‌دهنده داده‌های پرت (Outliers) باشد. این نمودار برای مقایسه توزیع داده‌ها در گروه‌های مختلف بسیار مفید است.

```
# Create a boxplot
sns.boxplot(x='input', y='output', data=df, width=0.2)
```

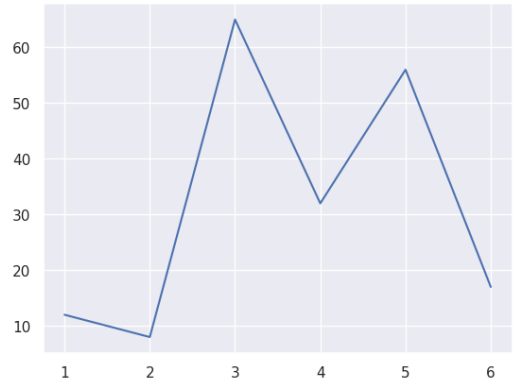


پایین ترین خط هر جعبه مقدار مینیموم، دومین خط هر جعبه میزان چارک اول (0.25) و خط وسط میانه یا چارک دوم (0.5) و خط سوم چارک سوم (0.75) را نمایش می دهد.

این که چارک اول برابر با 4120 باشد، به این معنی است که 0.25 درصد داده ها از عدد 4120 کوچک ترند.

برای اینکه چارتمون یه بک گراند زیبایی داشته باشه می تونیم از sns.set() قبل plot هامون استفاده کنیم.

```
sns.set()
x = [1, 2, 3, 4, 5, 6]
y = [12, 8, 65, 32, 56, 17]

plt.plot(x, y)
```




```
sns.set_style("dark")
```

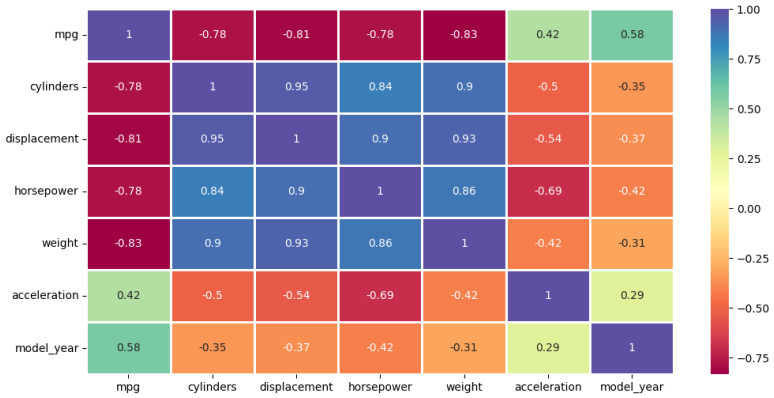
```
sns.set_style("whitegrid")
```

از darkgrid, whitegrid, dark, white, tick میتونیم در تابع set_style استفاده کنیم.

heatmap

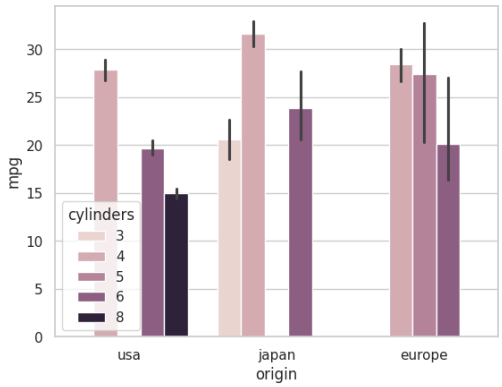
فرض کنید یک دیتاست داریم و میخواهیم نمودار heatmap انحراف از معیار ستون های آن را نمایش دهیم.

```
plt.figure(figsize = (12,6))

numcols = df.select_dtypes(include=np.number).columns
sns.heatmap(df[numcols].corr(), annot=True, linewidth = 2, cmap =
'Spectral')
```



Barplot

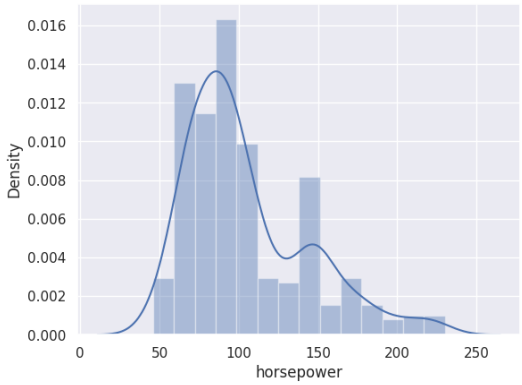
```
sns.set_style("whitegrid")
sns.barplot(x = 'origin', y = 'mpg', hue = "cylinders", data = df)
```



اسم ستون های مورد نظر رو توی x و y و hue دادیم، دیتافریم رو هم توی بخش data.

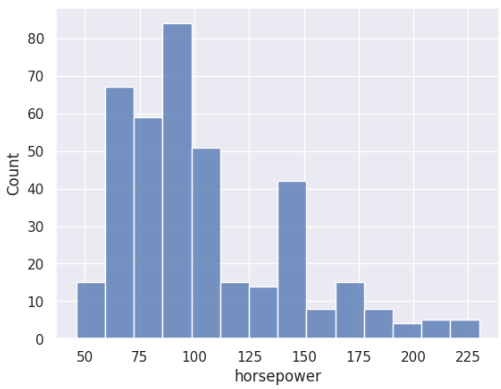
Distplot

```
sns.set_style("darkgrid")
sns.distplot(df['horsepower'])
```



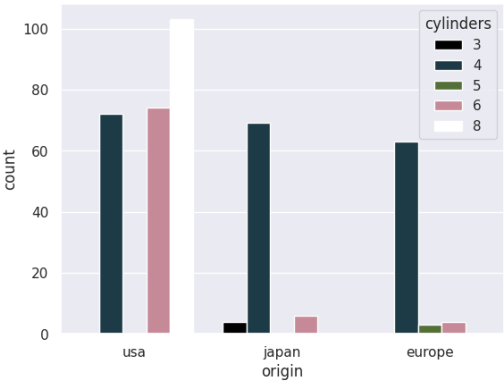
Histplot

```
sns.set_style("darkgrid")
sns.histplot(df['horsepower'])
```



Countplot

```
sns.set_style("darkgrid")
sns.countplot(x = "origin", hue='cylinders' ,data = df,
palette="cubehelix")
```



تحلیل توصیفی

روش های تحقیق توصیفی بسیار شبیه به توصیف وضعیت هستند. در این روش پیش‌بینی دقیق صورت نمی‌گیرد. همچنین علت و تاثیر تعیین نمی‌شود و در واقع آن چیز که موجود است، توصیف می‌شود. سه نوع اصلی از روش های توصیفی وجود دارد: روش مشاهده، روش مورد پژوهشی و روش نظرسنجی. هدف از پژوهش توصیفی بررسی مسائل و مشکلات فعلی از طریق فرآیند جمع آوری داده‌ها است. که پژوهشگر را قادر می سازد وضعیت را به طور کامل‌تر از آنچه که بدون استفاده از این روش امکان پذیر است توصیف کند.

تحلیل توصیفی در علم داده به بررسی خصوصیات اصلی داده‌ها می‌پردازد .این تحلیل با استفاده از شاخص‌های آماری مانند میانگین، میانه، مد و انحراف از معیار انجام می‌شود.

1. **درک داده‌ها** :ابتدا باید داده‌ها را بررسی کنیم و با ساختار آن‌ها آشنا شویم .این مرحله شامل بررسی نوع داده‌ها (عددی، طبقه‌ای یا ترتیبی) و تعداد مشاهدات و متغیرها می‌شود.
2. **پاکسازی داده‌ها** :در این مرحله، داده‌های ناقص یا خطا دار را پیدا و پاک یا جایگزین می‌کنیم. همچنین، داده‌های پرت و نویز را نیز شناسایی و در صورت نیاز حذف می‌کنیم.
3. **تجزیه و تحلیل داده‌ها** :در این مرحله، شاخص‌های آماری مانند میانگین، میانه، مد، واریانس، انحراف معیار، کوئارتایل‌ها (صدک ها)، بازه اطمینان و میزان همبستگی داده ها را محاسبه می‌کنیم، این شاخص‌ها به ما اطلاعات مفیدی در مورد توزیع داده‌ها می‌دهند.
4. **تصویرسازی داده‌ها** :با استفاده از نمودارها و گراف‌ها، می‌توانیم توزیع داده‌ها و روابط بین متغیرها را بررسی کنیم. نمودارهای میله‌ای، نمودارهای پراکندگی و جعبه‌ای از جمله نمودارهایی هستند که می‌توانند در تحلیل توصیفی مفید باشند.
5. **گزارش داده‌ها** :در نهایت، نتایج تحلیل را به صورت گزارش می‌نویسیم .این گزارش باید شامل تمامی نتایج محاسبات و تصویرسازی‌ها باشد و بتواند به خواننده درک کاملی از داده‌ها بدهد.

تحلیل توزیع داده ها

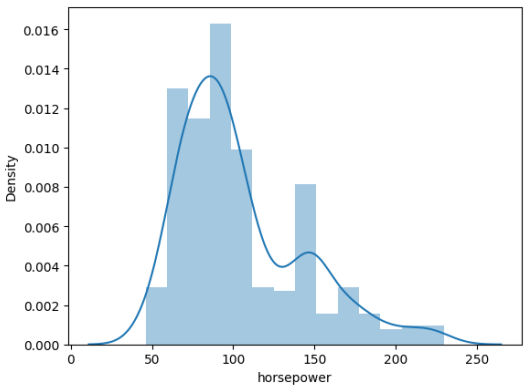
تحلیل توزیع داده‌ها یکی از مراحل مهم در فرایند تحلیل داده‌ها است .این فرایند به ما کمک می‌کند تا بفهمیم چگونه داده‌هایی که جمع‌آوری کرده‌ایم پراکنده شده‌اند.

یکی از مهم‌ترین توزیع‌های آماری، «توزیع نرمال» یا «تابع گاوسی» است. این توزیع یکی از مهمترین توزیع‌های احتمالی پیوسته در نظریه احتمالات است. توزیع نرمال به ما می‌گوید که پراکندگی و گستردگی داده‌هایی که جمع‌آوری کرده‌ایم چگونه است.

برای تحلیل داده‌ها، ابتدا باید هدف و اهداف کلیدی تجزیه و تحلیل داده‌های خود را مشخص کنید. سپس باید نوع تجزیه و تحلیل داده مورد استفاده را تعیین کنید.

در نهایت، تحلیل داده‌ها به ما اجازه می‌دهد تصمیمات را براساس شواهد قطعی و نه حدس‌وگمان اتخاذ کنیم. این علم به ما در گردآوری، منظم‌سازی و سر در آوردن از معانی پشت داده‌ها کمک می‌کند.

```
sns.distplot(df['horsepower'])
```



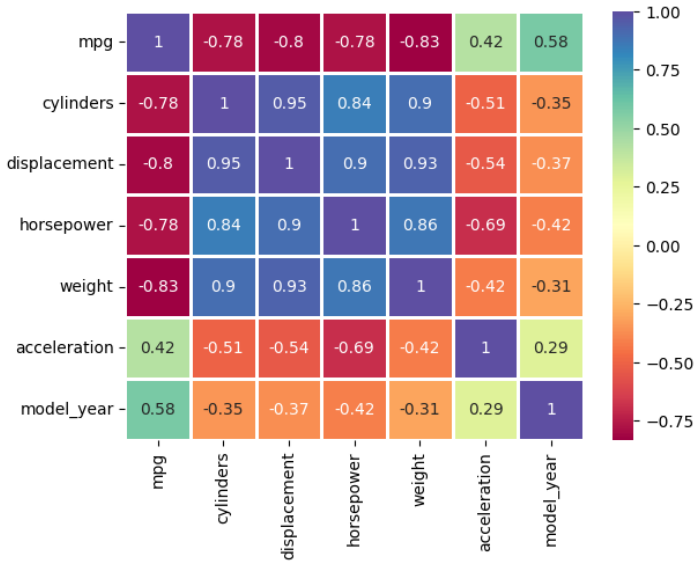
در نمودار بالا یک طرح از پراکندگی داده های ستون horsepower نمایش داده شده است.

تحلیل همبستگی

تحلیل همبستگی یکی از مراحل مهم در فرایند تحلیل داده‌ها است که به ما کمک می‌کند تا بفهمیم چگونه دو یا چند متغیر با یکدیگر ارتباط دارند. یکی از مهم‌ترین شاخص‌های همبستگی، «ضریب همبستگی یا Correlation Coefficients است. این شاخص نشان‌دهنده قدرت و جهت ارتباط بین دو متغیر است. برای مثال، اگر ضریب همبستگی بین دو متغیر مثبت باشد، بیانگر این است که با افزایش یکی، دیگری نیز افزایش می‌یابد .[1]اگر منفی باشد، بیانگر این است که با افزایش یکی، دیگری کاهش می‌یابد.

همچنین، می‌توان از «تحلیل همبستگی کانونی» یا «Canonical Correlation Analysis» برای بررسی ارتباط بین دو مجموعه از متغیرها استفاده کرد. این تحلیل به ما اجازه می‌دهد تا ببینیم چه چیزی بین این دو مجموعه مشترک است. ضریب همبستگی می‌تواند در محدوده 1- تا 1 باشد. مقدار 1 نشان‌دهنده ارتباط مستقیم کامل بین دو متغیر است، مقدار 1- نشان‌دهنده ارتباط معکوس کامل بین دو متغیر است و مقدار 0 نشان‌دهنده عدم وجود هرگونه ارتباطی بین دو متغیر است. در نهایت، باید توجه داشت که همبستگی بیانگر ارتباط علت و معلولی نیست، بلکه فقط معیاری برای نشان دادن میزان وابستگی بین دو متغیر محسوب می‌شود.

```
numcols = df.select_dtypes(include=np.number).columns
sns.heatmap(df[numcols].corr(), annot=True, linewidth = 2, cmap =
'Spectral')
```



در کتابخانه sklearn چندین تابع وجود دارد که می توانند ارتباط معنی دار بین دو ستون داده را بررسی کنند. در زیر به برخی از معروف ترین آنها اشاره می کنم:

1. **Pearson Correlation Coefficient ضریب همبستگی پیرسون** :این تابع در numpy به نام numpy.corrcoef وجود دارد و یک ضریب همبستگی بین -1 و 1 برمی گرداند. مقدار 1 نشان دهنده یک رابطه مستقیم کامل، -1 نشان دهنده یک رابطه معکوس کامل و 0 نشان دهنده عدم وجود رابطه است.
2. **Spearman Rank Correlation ضریب همبستگی رتبه ای اسپیرمن :** این تابع در scipy به نام scipy.stats.spearmanr وجود دارد و برای بررسی رابطه بین دو متغیر که رابطه غیر خطی دارند، مفید است.
3. **Chi-Square Test آزمون کای دو:** این تابع در scipy به نام scipy.stats.chi2_contingency وجود دارد و برای بررسی رابطه بین دو متغیر رده ای (categorical) استفاده می شود.

4. **ANOVA تحلیل واریانس :** این تابع در scipy به نام scipy.stats.f_oneway وجود دارد و برای بررسی تفاوت میانگین بین سه گروه یا بیشتر استفاده می شود.
5. **Mutual Information اطلاعات متقابل:** این تابع در sklearnبه نام sklearn.feature_selection.mutual_info_classif وجود دارد و میزان اطلاعات مشترک بین دو متغیر را محاسبه می کند.

این توابع می توانند به شما کمک کنند تا بفهمید که آیا رابطه معنی داری بین دو ستون داده وجود دارد یا خیر. اما لازم است توجه داشته باشید که همیشه باید نوع داده ها و نوع رابطه مورد نظر را در نظر گرفته و تابع مناسب را انتخاب کنید. همچنین، همیشه باید به معنای آماری این توابع و محدودیت های آنها توجه کرد.

آزمون فرضیه

آزمون فرضیه یک روش آماری است که برای تصمیم گیری در مورد فرضیه استفاده می شود. این فرضیه می تواند در مورد پارامترهای جمعیت باشد. به عنوان مثال، میانگین یا واریانس یک جمعیت خاص.

در آزمون فرضیه، دو نوع فرضیه وجود دارد:

1. **فرض صفر** :**(H0)** این فرضیه ادعا می کند که هیچ ارتباط معنی داری بین داده ها وجود ندارد. به عنوان مثال، اگر ما دو گروه داریم و می خواهیم ببینیم آیا میانگین دو گروه متفاوت است یا خیر، فرض صفر می گوید که میانگین دو گروه برابر نیست.
2. **فرض جایگزین** :**(H1)** این فرضیه ادعا می کند که اربتباط معنی داری بین داده ها وجود دارد. در مثال قبلی، فرض جایگزین می گوید که میانگین دو گروه برابر است.

برای انجام آزمون فرضیه، ما ابتدا فرض صفر را انتخاب می کنیم و سپس داده ها را بررسی می کنیم. اگر داده ها به شدت با فرض صفر مغایرت داشته باشند، ما فرض صفر را رد می کنیم و فرض جایگزین را قبول می کنیم.

به عنوان مثال، فرض کنید که می خواهیم بررسی کنیم آیا دانش آموزان دختر در یک کلاس ریاضی بهتر از دانش آموزان پسر عمل می کنند یا خیر. در اینجا، فرض صفر می گوید که نمرات میانگین دختران و پسران برابر نیست (یعنی جنسیت تأثیری بر نمرات دارد) و فرض جایگزین می گوید که نمرات میانگین دختران و پسران برابر است (یعنی جنسیت تأثیری بر نمرات ندارد). سپس ما داده ها را جمع آوری و آزمون فرضیه را انجام می دهیم تا ببینیم کدام فرضیه را باید قبول کنیم.

در آزمون فرضیه P_value احتمال رخداد فرضیه صفر است، هر چه مقدار این مقدار کمتر باشد، احتمال رخداد فرض صفر کمتر خواهد بود. معمولا مقدار احتمال 5 درصد برای کارهای تحقیقاتی و 1 درصد برای ابزار دقیق به کار می رود. هرچه این مقدار کمتر باشد به این معنی است که فرض صفر با قدرت بیشتری رد می شود، و فرض جایگزین مورد قبول است.

در آزمون فرضیه Alpha (همون 5 درصدر در جمله قبل) مقداری است برای سنجش P_value، اگر مقدار P_value از Alpha کمتر باشد، فرض صفر رد می شود و اگر بیشتر باشد فرض صفر مورد قبول واقع می شود.

یک نمونه از اجرای آزمون فرضیه برای دو ستون عددی پیوسته از دیتاست.

```
# from scipy import stats

# H0: There is not a correlation.
# H1: There is a significant correlation.

# Assuming df is your DataFrame and 'col1' and 'col2' are the columns
spearman_corr, p_value = stats.spearmanr(df['cylinders'], df['mpg'])

print(f"Spearman Rank Correlation: {spearman_corr}")
print(f"P-value: {p_value}")

# Hypothesis testing
alpha = 0.05  # Or whatever significance level you want
if p_value < alpha:
    print("We reject the null hypothesis and accept the alternative hypothesis. There is a significant correlation.")
else:
    print("We fail to reject the null hypothesis. There is no significant correlation.")
```

```
Spearman Rank Correlation: -0.8218644914450967
P-value: 8.284686810807197e-99
We reject the null hypothesis and accept the alternative hypothesis. There is a significant correlation.
```

در مثال بالا کاملا مشخص است که ارتباط معنی داری بین داده ها وجود دارد و فرض صفر که می گفت ارتباطی وجود ندارد رد شده است.

یادگیری ماشین

یادگیری ماشین به انگلیسی (Machine learning) به اختصار **ML** مطالعهٔ الگوریتم‌ها و مدل‌های آماری مورد استفادهٔ سیستم‌های کامپیوتری است که به‌جای استفاده از دستورالعمل‌های واضح، از الگوها و استنباط برای انجام وظایف استفاده می‌کنند. یادگیری ماشینی علمی است که باعث می‌شود رایانه‌ها بدون نیاز به یک برنامه صریح در مورد یک موضوع خاص یاد بگیرند. به عنوان زیر مجموعه‌ای از هوش مصنوعی، الگوریتم‌های یادگیری ماشینی یک مدل ریاضی بر اساس داده‌های نمونه یا داده‌های آموزش به منظور پیش‌بینی یا تصمیم‌گیری بدون برنامه‌ریزی آشکار، ایجاد می‌کنند.

یادگیری **تحت نظارت** به انگلسیس **Supervised Learning** یک روش عمومی در یادگیری ماشینی است که در آن به یک سیستم، مجموعه‌ای از جفت‌های ورودی – خروجی ارائه شده و سیستم تلاش می‌کند تا تابعی از ورودی به خروجی را فرا گیرد. یادگیری تحت نظارت نیازمند تعدادی داده ورودی به منظور آموزش سیستم است. یادگیری تحت نظارت خود به دو دسته تقسیم می‌شود :**رگرسیون و طبقه‌بندی**.

رگرسیون آن دسته از مسائل هستند که خروجی یک عدد پیوسته یا یک سری اعداد پیوسته هستند مانند پیش‌بینی قیمت خانه بر اساس اطلاعاتی مانند مساحت، تعداد اتاق خواب‌ها، و غیره و دسته **طبقه‌بندی** به آن دسته از مسائل گفته می‌شود که خروجی یک عضو از یک مجموعه باشد مانند پیش‌بینی اینکه یک ایمیل هرزنامه هست یا خیر یا پیش‌بینی نوع بیماری یک فرد از میان ۱۰ بیماری از پیش تعریف شده.

یادگیری **بدون نظارت** به انگلیسی **UnSupervised Learning** یکی از انواع یادگیری در یادگیری ماشین است. اگر یادگیری بر روی داده‌های بدون برچسب و برای یافتن الگوهای پنهان در این داده‌ها انجام شود، یادگیری بدون نظارت خواهد بود از انواع یادگیری بدون نظارت می‌توان به الگوریتم‌های **خوشه‌بندی** (Clustering) تخصیص پنهان دیریکله (LDA) و جاسازی لغات (Word Embedding) اشاره کرد.

از یادگیری نظارت نشده در دنیای امروز می‌توان مثال‌های متعددی زد. یکی از پُرکاربردترین آن‌ها پیشنهادهایی است که به کاربران در شبکه‌های اجتماعی داده می‌شود. به عنوان مثال در اینستاگرام داده‌های زیادی از هر کاربر از جمله علایق شخصی، کسانی که دنبال می‌کند، دنبال‌کنندگان او وجود دارد. اینستاگرام براساس این داده‌ها، ویژگی‌های کابران را تعیین کرده و آن‌ها را خوشه‌بندی می‌کند. در نهایت با توجه به خوشه‌ای که کاربر درون آن قرار گرفته‌است، پیشنهادهای متعددی به وی در جهت درگیر کردن بیشتر او با این شبکه اجتماعی می‌دهد.

یادگیری تقویتی، هدف یادگیری تقویتی بخشی که از یادگیری ماشینی است این است که چگونه عامل‌های نرم‌افزاری، باید یک عمل را مناسب محیط انتخاب کنند تا پاداش بهینه بیشینه شود. این رشته به دلیل کلی بودن، در بسیاری از رشته‌های دیگر از جمله نظریه بازی، تئوری کنترل، تحقیق در عملیات، تئوری اطلاعات، بهینه‌سازی مبتنی بر شبیه‌سازی، سیستم‌های چند عامل، هوشمند جمعی، آمار و الگوریتم‌های ژنتیکی مورد مطالعه قرار می‌گیرد. در یادگیری ماشین، محیط به‌طور معمول به عنوان یک فرایند تصمیم‌گیری مارکوف (MDP) معرفی می‌شود. بسیاری از الگوریتم‌های یادگیری تقویتی از تکنیک‌های برنامه‌نویسی پویا استفاده می‌کنند.

در الگوریتم‌های یادگیری تقویتی، فرضیه مبتنی بر دانش یک مدل دقیق ریاضی از MDP نیست، و هنگامی که مدل‌های دقیق غیرقابل دسترسی هستند مورد استفاده قرار می‌گیرد. الگوریتم‌های یادگیری تقویتی در وسایل نقلیه خودران یا در یادگیری بازی در برابر حریف انسانی استفاده می‌شود.

Regression

رگرسيون يعني بازگشت. يعني پيش بيني و بيان تغييرات يک متغير بر اساس اطلاعات متغير ديگر.

مثال: رابطه بين قد و وزن انسانها را در نظر بگيريد. همه مي دانيم که اين رابطه يک رابطه مستقيم رياضي و صد درصدي نيست که لزوما هر که قد بلندتري داشته باشد وزن بيشتري داشته باشد، اما مي توان گفت که با احتمال قابل قبولي افراد با قد بلندتر، وزن بيشتري نيز دارند. در اينجا پيش بيني وزن از روي قد و بيان ارتباط بين اين متغير با روش آماري رگرسيون خطي صورت مي پذيرد که اين رابطه را به صورت کمي به ما نشان مي دهد.

معمولاً زمانی می‌توان از رگرسیون استفاده کرد که یک همبستگی بین متغیرهای مستقل و وابسته وجود داشته باشد. این همبستگی را می‌توان به عنوان مثال از ضریب همبستگی پیرسن که عددی در بازه[1 ,1-] است به‌دست‌آورد. ضریب همبستگی مثبت به این معنی است که با افزایش یک متغیر، متغیر دیگر هم افزایش می‌یابد و بالعکس.

انواع مختلفی از رگرسیون یا Regression وجود دارد.

- رگرسیون خطی ساده (Simple Linear Regression)
- رگرسیون خطی چندگانه (Multiple Linear Regression)
- رگرسیون لجستیک (Logistic Regression)
- رگرسیون پلینومی (Polynomial Regression)
- و انواع دیگر ...

قبل از اینکه بریم سراغ رگرسیون یکم راجب شیوه محاسبه خطا در مدل های رگرسیون بخونیم.

- **MAE (Mean Absolute Error)**: میانگین خطای مطلق، که با فرمول

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

محاسبه می‌شود، جایی که (Y_i) مقادیر واقعی و (X_i) پیش‌بینی‌های مدل هستند MAE .مقدار متوسط اختلاف بین مقادیر واقعی و پیش‌بینی‌ها را نشان می‌دهد و به خطاهای بزرگتر و کوچکتر وزن یکسانی می‌دهد.

- **MSE (Mean Squared Error)**: میانگین مربعات خطا، که با فرمول

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

محاسبه می‌شود. MSE خطاهای بزرگتر را با وزن بیشتری مجازات می‌کند، زیرا خطاها به توان دو رسیده‌اند.

- **RMSE (Root Mean Squared Error)**: جذر میانگین مربعات خطا، که با فرمول

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}}$$

محاسبه می‌شود RMSE .مشابه MSE است اما با این تفاوت که واحد خطا را به واحد اصلی متغیر وابسته برمی‌گرداند و بنابراین تفسیر آن آسان‌تر است.

- **R2 Score (Coefficient of Determination)**: ضریب تعیین، که میزان تغییرات متغیر وابسته را که توسط مدل توضیح داده شده است، نشان می‌دهد. مقدار R2 بین 0 تا 1 متغیر است، جایی که 1 نشان‌دهنده این است که مدل تمام تغییرات را توضیح می‌دهد و 0 به این معنی است که مدل هیچ تغییری را توضیح نمی‌دهد. فرمول آن به صورت زیر است :

$$1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}.$$

که در آن $\bar{y}$ میانگین مقادیر واقعی است.

این معیارها به ما کمک می‌کنند تا درک بهتری از عملکرد مدل‌های رگرسیونی داشته باشیم و تصمیم‌گیری‌های آگاهانه‌تری در مورد بهبود یا انتخاب مدل‌ها انجام دهیم.

Simple Linear Refression

رگرسیون خطی ساده میزان اثر یک متغیر مستقل بر یک متغیر وابسته را می‌سنجد و همبستگی رابطهٔ بین آن‌ها را مورد سنجش قرار می‌دهد.

فرم مدل رگرسیون خطی ساده به صورت زیر است:

$$Y=\beta_0+\beta_1 X+\epsilon$$

همانطور که دیده می‌شود این رابطه، معادله یک خط است که جمله خطا یا همان ϵ به آن اضافه شده. پارامترهای این مدل خطی عرض از مبدا (β_0) و شیب خط (β_1) است. **شیب خط** در حالت رگرسیون خطی ساده، نشان می‌دهد که میزان حساسیت متغیر وابسته به متغیر مستقل چقدر است. به این معنی که با افزایش یک واحد به مقدار متغیر مستقل چه میزان متغیر وابسته تغییر خواهد کرد. **عرض از مبدا** نیز بیانگر مقداری از متغیر وابسته است که به ازاء مقدار متغیر مستقل برابر با صفر محاسبه می‌شود. به شکل دیگر می‌توان مقدار ثابت یا عرض از مبدا را مقدار متوسط متغیر وابسته به ازاء حذف متغیر مستقل در نظر گرفت.

گاهی مدل رگرسیونی را بدون عرض از مبدا در نظر می‌گیرند و $\beta_0=0$ محسوب می‌کنند. این کار به این معنی است که با صفر شدن مقدار متغیر مستقل، مقدار متغیر وابسته نیز باید صفر در نظر گرفته شود. زمانی که محقق مطمئن باشد که که خط رگرسیون باید از مبدا مختصات عبور کند، این گونه مدل در نظر گرفته می‌شود. فرم مدل رگرسیونی در این حالت به صورت زیر است:

$$Y=\beta_1 X+\epsilon$$

مثال:

فرض کنید یک ستون به عنوان سن خوردرو و یک ستون به نام mpg که میزان سوزاندن سوخت آن خودرو است را داریم، می خواهیم میزان mpg را از روی سن خودرو بدست آوریم. Intercept عرض از مبدا و Coefficient مقدار شیب خط ماست.

```
# from sklearn.linear_model import LinearRegression
# from sklearn.model_selection import train_test_split
# import pandas as pd

X = df[['age']]
y = df['mpg']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()

model.fit(X_train, y_train)

print("Intercept: ", model.intercept_)
print("Coefficient: ", model.coef_)
```

نمایش معادله رگرسیون

```
print("Equation of the model: mpg = {} * age + {}".format(model.coef_[0],
model.intercept_))
```

```
predictions = model.predict(X_test)
```

محاسبه دقت مدل

```
# from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Calculate MAE
mae_value = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error (MAE): {mae_value:.2f}")

# Calculate MSE
mse_value = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error (MSE): {mse_value:.2f}")

# Calculate RMSE
RMSE = np.sqrt(mse_value)
print("R Mean Squared Error (RMSE): ", RMSE)

# Calculate R2 score
r2_value = r2_score(y_test, predictions)
print(f"R-squared (R2) Score: {r2_value:.2f}")
```

Multiple Linear Regression

رگرسیون خطی چندگانه (Multiple linear regression) با نام متداول MLR که به سادگی به عنوان رگرسیون چندگانه نیز شناخته می شود، یک تکنیک آماری است که از چندین متغیر توضیحی برای پیش بینی نتیجه یک متغیر پاسخ استفاده می کند. هدف رگرسیون خطی چندگانه مدل سازی رابطه خطی بین متغیرهای مستقل و متغیر وابسته است .

$$y = \beta_0 + \beta_1 X_1 + \beta_2 x_2 + \cdots + \beta_i X_i + \epsilon$$

در معادله فوق β_0 عرض از مبدا و $\beta_1 \dots \beta_i$ شیب خط های ما هستند، $X1 \dots Xi$ متغیر های مستقل و یا ورودی های ما هستند، همچنین Y متغیر وابسته و خرووجی معادله رگرسیون، و آخرین پارامتر یعنی ϵ یا اپسیلون میزان خطای محاسبه شده ما هست.

مثال: فرض کنید که با 4 پارامتر یا متغیر مستقل به نام های ['cylinders', 'displacement', 'horsepower', 'acceleration'] میخواهیم میزان یک متغیر وابسته به نام mpg را پیش بینی کنیم.

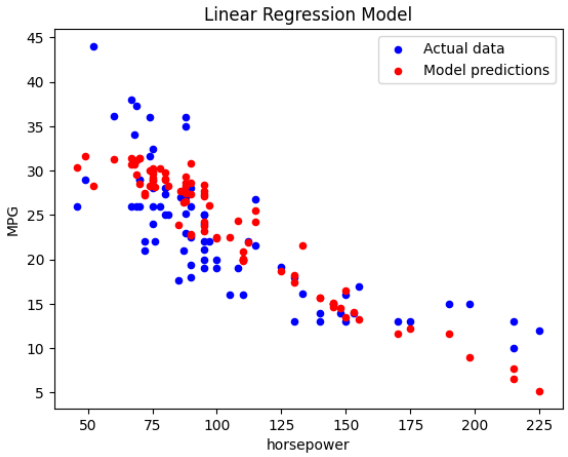
```
# from sklearn.linear_model import LinearRegression
# from sklearn.model_selection import train_test_split

x = df[['cylinders', 'displacement', 'horsepower', 'acceleration']]
y = df['mpg']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

model = LinearRegression()

model.fit(x_train, y_train)
```

```
pred = model.predict(x_test)
```

```
# from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Calculate MAE
mae_value = mean_absolute_error(y_test, pred)
print(f"Mean Absolute Error (MAE): {mae_value:.2f}")

# Calculate MSE
mse_value = mean_squared_error(y_test, pred)
print(f"Mean Squared Error (MSE): {mse_value:.2f}")

# Calculate RMSE
RMSE = np.sqrt(mse_value)
print("R Mean Squared Error (RMSE): ", RMSE)

# Calculate R2 score
r2_value = r2_score(y_test, pred)
print(f"R-squared (R2) Score: {r2_value:.2f}")
```

نمایش نمودار y های واقعی با y های پیشبینی شده

```
y_pred = model.predict(x_test)

plt.scatter(x_test['horsepower'], y_test, color='blue', label='Actual
data', s=20)

plt.scatter(x_test['horsepower'], y_pred, color='red', label='Model
predictions', s=20)

plt.xlabel('horsepower')
plt.ylabel('MPG')
plt.title('Linear Regression Model')
plt.legend()

plt.show()
```

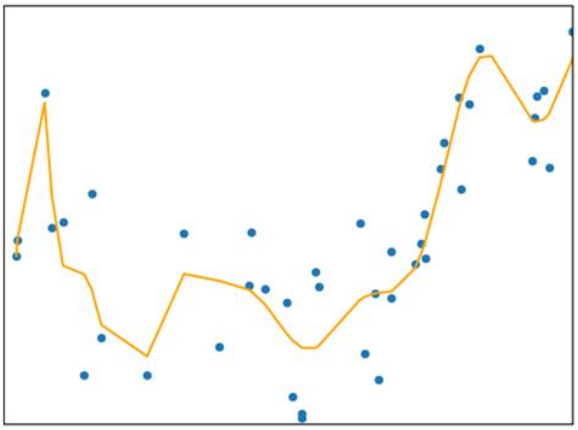


Logistic Regression

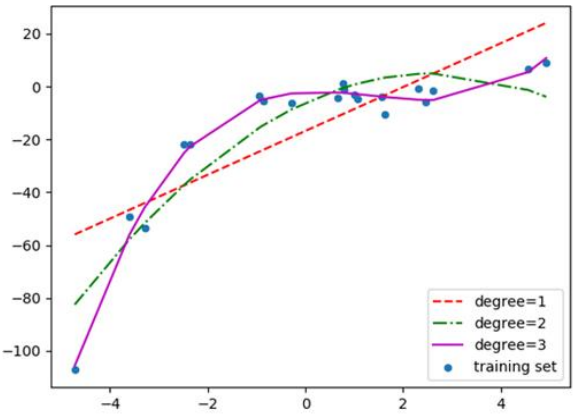
رگرسیون لاجستیک یک مدل Classification هست، پس توی قسمت های بعدی راجبش صحبت می کنیم

Polynomial Regression

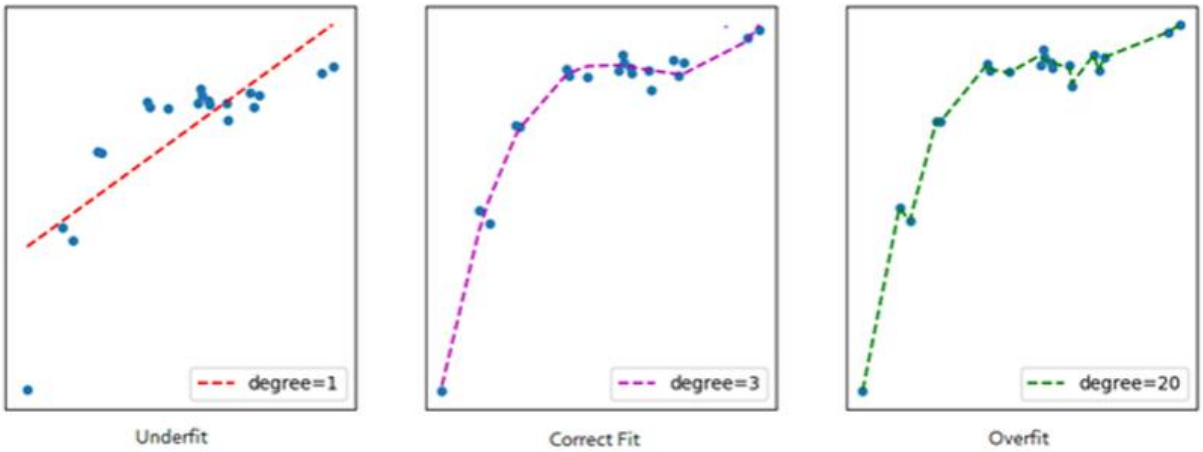
رگرسیون چند جمله ای شکلی از تحلیل رگرسیون است که در آن رابطه بین متغیرهای مستقل و متغیرهای وابسته در چند جمله ای درجهn ام مدل شده است. رگرسیون چند جمله ای یک مورد خاص از رگرسیون خطی است که در آن معادله چند جمله ای را بر روی داده ها با رابطه منحنی بین متغیرهای وابسته و مستقل قرار می دهید.



تفاوت درجه های مختلف در رگرسیون را در تصویر زیر می توانید مشاهده کنید.



بالا بردن درجه معادله همیشه باعث بالاتر رفتن دقت مدل نمیشود، بالا بردن درجه معادله بیش از اندازه میتواند باعث overfiting شود. Overfit یعنی مدل ما داده های train رو حفظ می‌کنه و در نتیجه نمیتونه روی داده های تست به دقت خوبی برسه.



مثال:

```
# from sklearn.preprocessing import PolynomialFeatures
# from sklearn.pipeline import make_pipeline
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LinearRegression

x = df[['cylinders', 'displacement', 'horsepower', 'acceleration']]
y = df['mpg']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

polynomial_model = make_pipeline(PolynomialFeatures(degree=5),
LinearRegression())

polynomial_model.fit(x_train, y_train)

y_pred = polynomial_model.predict(x_test)

# Calculate MAE
```

```
mae_value = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae_value:.2f}")

# Calculate MSE
mse_value = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse_value:.2f}")

# Calculate RMSE
RMSE = np.sqrt(mse_value)
print("R Mean Squared Error (RMSE): ", RMSE)

# Calculate R2 score
r2_value = r2_score(y_test, pred)
print(f"R-squared (R2) Score: {r2_value:.2f}")
```

یک کد برای تست درجه های مختلف روی داده

```
# from sklearn.preprocessing import PolynomialFeatures
# from sklearn.pipeline import make_pipeline
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LinearRegression

x = df[['cylinders', 'displacement', 'horsepower', 'acceleration']]
y = df['mpg']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

best_d = 1
best_r = -float('inf')
for i in range(1, 20):
    polynomial_model = make_pipeline(PolynomialFeatures(degree=i),
LinearRegression())

    polynomial_model.fit(x_train, y_train)

    y_pred = polynomial_model.predict(x_test)

    r2_value = r2_score(y_test, y_pred)

    if r2_value > best_r:
        best_d = i
        best_r = r2_value

print("Best degree:", best_d)
```

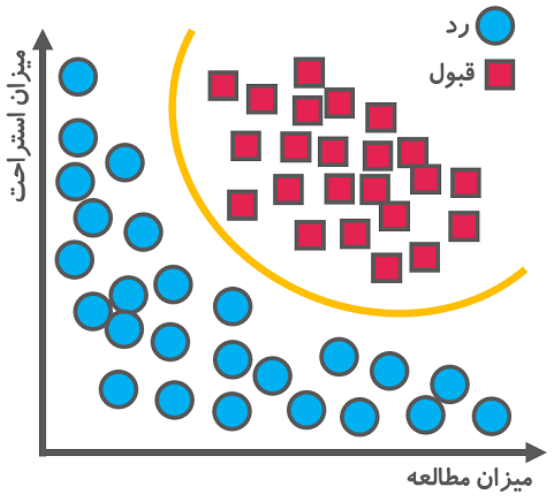
طبقه بندی

در یادگیری ماشین به فرایند تشخیص، درک و دسته‌بندی اشیا و ایده‌ها به دسته‌های موجود و تعریف شده به عنوان زیردسته‌ای از جامعه فعلی، طبقه‌بندی می‌گویند. به کمک مجموعه داده‌های آموزشی از پیش طبقه‌بندی شده، برنامه‌های طبقه بندی در یادگیری ماشین، آموزش می‌بینند و با استفاده از گستره‌ی وسیعی از الگوریتم ها مجموعه داده‌ها (Datasets) که در آینده با آن به عنوان مسئله روبه‌رو خواهند شد را می‌توانند به دسته‌های مرتبط و معنی‌دار، طبقه‌بندی کنند. الگوریتم‌های استفاده شده برای طبقه بندی در یادگیری ماشین از داده‌های آموزشی ورودی، برای پیش‌بینی احتمال عضویت داده‌ها در یکی از دسته‌بندی‌های از پیش‌تعیین شده استفاده می‌کنند.

انواع طبقه بندی

1. طبقه بندی دودویی :کل داده ها فقط به دو دسته تقسیم می شوند.
2. طبقه بندی چند کلاسه :داده ها به دسته هایی بیش تر از دوتا تقسیم می شوند.

در تصویر زیر یک طبقه بندی دو کلاسه را میبینید که با میزان ساعت مطالعه دانش آموزان احتمال قبولی و رد شدن آن ها در امتحاناتشان را مشخص کرده است.



چندتا از الگوریتم های طبقه بندی

1. رگرسیون لاجستیک Logistic Regression

این الگوریتم یک الگوریتم دسته بندی یا طبقه بندی دو دویی است، یعنی دیتای ما فقط باید دو کلاس داشته باشد. برای این منظور، رگرسیون لجستیک از تابعی به‌نام «سیگموئید» (Sigmoid) استفاده کرده و احتمال تعلق داشتن ورودی به هر کدام از کلاس‌ها را محاسبه می‌کند.

2. الگوریتم بیز ساده Naive Bayes

مدل‌های بیز ساده (Naive Bayes) زیرمجموعه‌ای از مدل‌های خطی هستند که فرض می‌کنند ویژگی‌های ورودی نسبت به یک‌دیگر مستقل هستند. یعنی با بررسی هر ویژگی به‌صورت جداگانه و با استفاده از قضیه بیز (Bayes Theorem)، احتمال یک برچسب کلاسی برای نمونه ورودی محاسبه می‌شود. متفاوت با اغلب روش های یادگیری ماشین که وابستگی زیادی به مجموعه‌داده‌های بزرگ دارند، الگوریتم بیز ساده حتی با مجموعه‌داده‌های کوچک نیز عملکرد به نسبت قابل قبولی از خود به جا می‌گذارد.

3. الگوریتم درخت تصمیم Decision Tree

از الگوریتم «درخت تصمیم» (Decision Tree) به عنوان راه‌حلی رایج برای نمایش نتایج یک تصمیم یا رفتار در غالب احتمالات یاد می‌شود. به‌طور خلاصه، درخت تصمیم نمایشی سلسله مراتبی از فضای خروجی است که در آن هر «گره» (Node) نماد یک انتخاب یا عمل و «برگ‌ها» (Leaves) همان برچسب‌ها هستند.

4. الگوریتم جنگل تصادفی Random Forest

همان‌طور که از اسمش پیداست، «جنگل تصادفی» (Random Forest) مجموعه‌ای از چند درخت تصمیم است. الگوریتم جنگل تصادفی، شیوه رایجی از روش‌های ترکیبی با نتایجی تجمعی از چندین مدل پیش‌بینی کننده است. تکنیک Bagging یکی از روش‌های مورد استفاده در جنگل تصادفی است که ابتدا هر درخت را با نمونه‌هایی تصادفی از مجموعه‌داده اصلی آموزش داده و سپس کلاسی انتخاب می‌شود که بیشتری تکرار را در میان گره‌های برگ داشته باشد. در مقایسه با درخت تصمیم، الگوریتم جنگل تصادفی قابلیت عمومی‌سازی بیشتر و تفسیرپذیری کمتری دارد.

5. الگوریتم ماشین بردار پشته ای Support Vector Machine (SVM)

به گروهی از الگوریتم های طبقه بندی که داده‌های شما را به یک فضای تصمیم خطی منتقل می‌کنند، «ماشین بردار پشتیبان» (Support Vector Machine | SVM) گفته می‌شود. این الگوریتم، داده‌ها را بر اساس مرزی با عنوان «ابَرصفحه» (Hyperplane) در دو کلاس مثبت و منفی طبقه‌بندی می‌کند.

6. الگوریتم k-نزدیک ترین همسایه (K-Nearest Neighbour)

می‌توانید به الگوریتم «K-نزدیک‌ترین همسایه» (K-Nearest Neighbour | KNN) به عنوان نمایی از نقاط داده در فضایی n بعدی نگاه کنید که n در واقع همان تعداد ویژگی‌های ورودی است. نحوه کار الگوریتم KNN به این صورت است که فاصله میان هر نمونه با سایر نقاط داده را محاسبه کرده و سپس به نزدیک‌ترین و دورترین نمونه‌ها، برچسب «همسایه» یا «غیر همسایه» می‌دهد.

مثال: فرض کنید یک دیتاست با سه دسته مختلف داشته باشیم

```
['setosa' 'versicolor' 'virginica']
```

	sepal_length	sepal_width	petal_length	petal_width	species
42	4.4	3.2	1.3	0.2	setosa
66	5.6	3.0	4.5	1.5	versicolor
89	5.5	2.5	4.0	1.3	versicolor
114	5.8	2.8	5.1	2.4	virginica
20	5.4	3.4	1.7	0.2	setosa

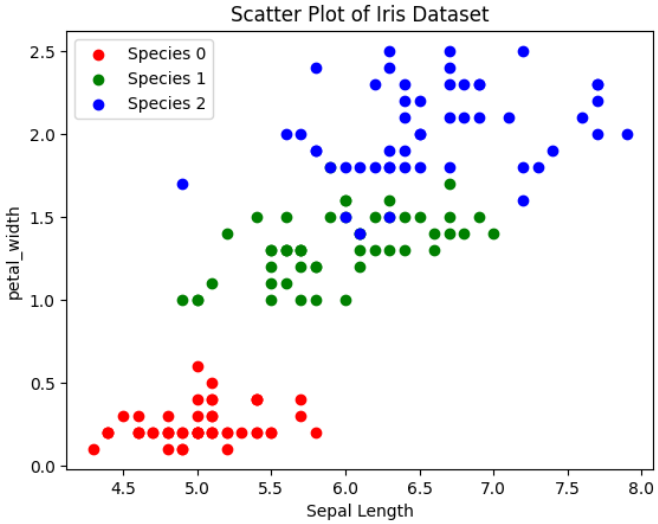
باید برای اول کار دیتای متنیمون روبه دیتای عددی تبدیل کنیم.

```
map = {'setosa' : 0, 'versicolor' : 1, 'virginica' : 2}
df['species'] = df['species'].map(map)
```

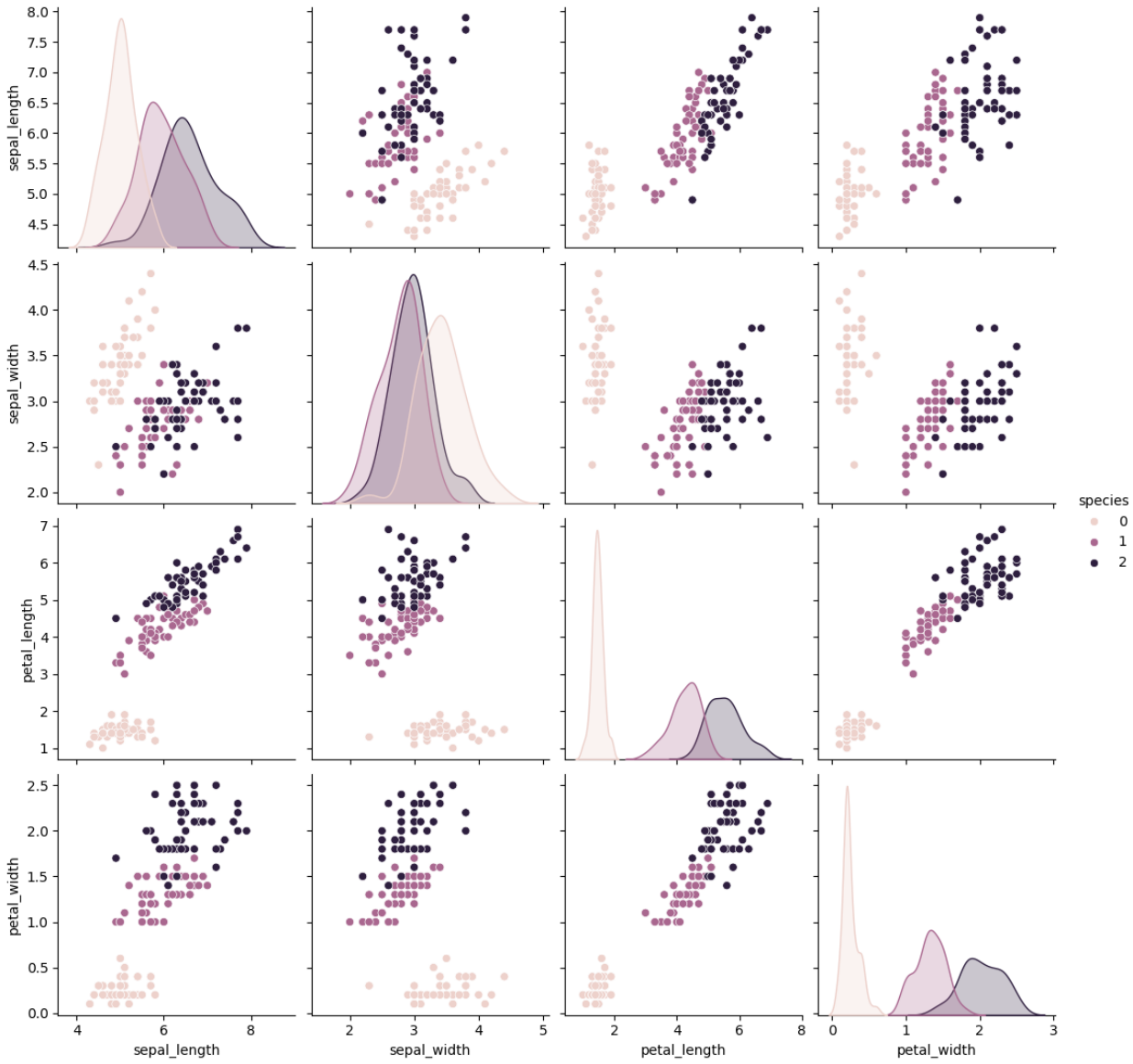
	sepal_length	sepal_width	petal_length	petal_width	species
21	5.1	3.7	1.5	0.4	0
92	5.8	2.6	4.0	1.2	1
23	5.1	3.3	1.7	0.5	0
100	6.3	3.3	6.0	2.5	2
47	4.6	3.2	1.4	0.2	0

یک نمایش برای دسته هامون با انتخاب دوتا از ستون ها ببینیم

```
colors = ['red', 'green', 'blue']
species = df['species'].unique()

for specie in species:
    subset = df[df['species'] == specie]
    plt.scatter(subset['sepal_length'], subset['petal_width'],
label=f'Species {specie}', color=colors[specie])

plt.title('Scatter Plot of Iris Dataset')
plt.xlabel('Sepal Length')
plt.ylabel('petal_width')
plt.legend()

plt.show()
```



```
sns.pairplot(df, hue='species', height = 2.75)
```

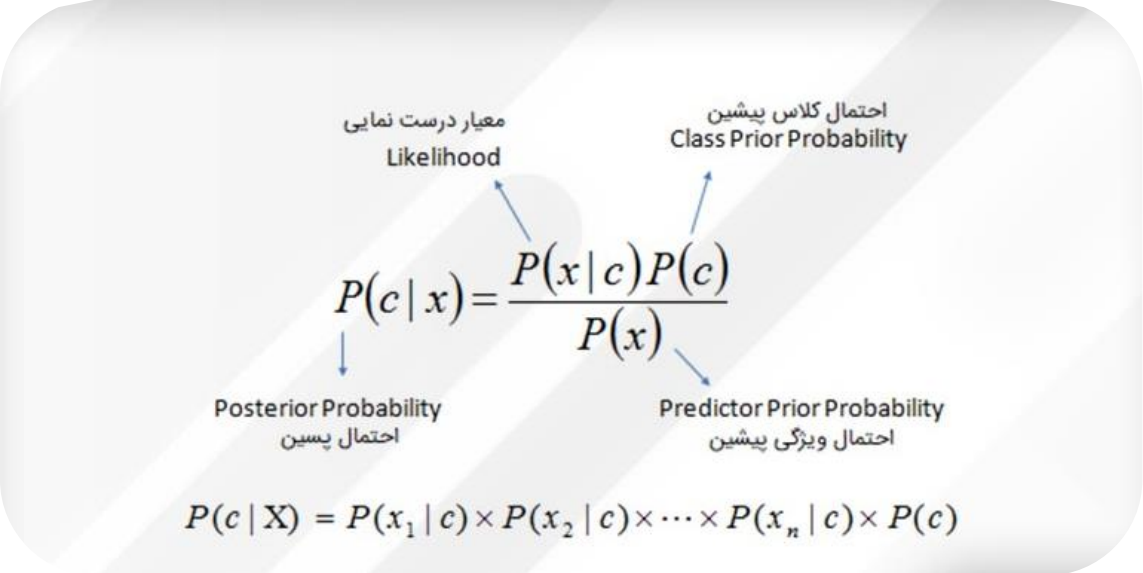


الگوریتم بیز ساده (Naive bayes)

الگوریتم Naive Bayes با یادگیری احتمال شرطی هر ویژگی داده شده در هر کلاس در داده‌های آموزشی کار می‌کند. هنگامی که داده‌های جدید و دیده نشده به الگوریتم ارائه می‌شود، الگوریتم احتمال هر کلاس را بر اساس ویژگی‌های مشاهده شده محاسبه می‌کند و کلاسی را که بیشترین احتمال را دارد به عنوان کلاس پیش‌بینی شده برای آن داده انتخاب می‌کند. با وجود سادگی و فرضیات گفته شده، مشخص شده است که Naive Bayes در طیف گسترده‌ای از وظایف طبقه‌بندی به خوبی عمل می‌کند، به خصوص زمانی که میزان داده‌های آموزشی در مقایسه با تعداد ویژگی‌ها کم باشد.

فرض کنید در حال قدم‌زدن در یک پارک هستید و جسمی قرمز در مقابل خود مشاهده می‌کنید. این جسم قرمز می‌تواند چوب‌دستی، خرگوش یا توپ باشد. شما قطعاً فرض می‌کنید که آن جسم یک توپ است. چرا؟ بیایید تصور کنیم که در حال ساخت ماشینی هستیم که وظیفه‌اش طبقه‌بندی یک جسم میان سه گروه چوب‌دستی، توپ و خرگوش است. در ابتدا احتمالاً به این فکر می‌کنیم که ماشینی که می‌سازیم بتواند ویژگی‌های جسم را شناسایی کند و سپس آن را با یکی از گروه‌هایی که می‌خواهیم در آن‌ها طبقه‌بندی کنیم مطابقت دهد.

به‌طوری که مثلاً اگر جسم دایره‌ای شکل باشد، پیش‌بینی کند که یک توپ است یا اگر آن جسم یک موجود زنده باشد، آن‌گاه پیش‌بینی ماشین ما خرگوش باشد، یا در مثالی که داشتیم، اگر جسم قرمز باشد، به‌احتمال زیاد آن را توپ در نظر بگیرد. چرا؟ زیرا از دوره‌ی کودکی ما توپ‌های قرمز زیادی دیده‌ایم، اما یک خرگوش قرمز یا یک چوب‌دستی قرمز در نظر ما بسیار بعید است. بنابراین در مثالی که داریم ویژگی ما رنگ قرمز است که آن را با هر سه گروه مطابقت می‌دهیم و می‌بینیم که احتمال اینکه این جسم یک توپ باشد خیلی بیشتر است.



- C نشان‌دهنده‌ی کلاس مدنظر است؛ مثلاً در مثالی که داشتیم کلاس‌ها خرگوش و چوب‌دستی و توپ هستند.
- X نشان‌دهنده‌ی ویژگی‌هاست که هر یک به‌طور جداگانه باید محاسبه شوند.
- P(C | X) احتمال پسین (Posterior) کلاس C با داشتن پیش‌بینی‌کننده‌ی (ویژگی) X است.
- P(c) احتمال کلاس است.
- P(X | C) معیار درست‌نمایی (Likelihood) است که احتمال پیش‌بینی‌کننده‌ی X با داشتن کلاس C را نشان می‌دهد.
- P(X) احتمال پیشین (Prior) پیش‌بینی‌کننده‌ی X است.

سه نوع اصلی بیز ساده وجود دارد:

- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Bernoulli Naive Bayes

Gaussian Naive Bayes

Gaussian Naive Bayes برای دسته‌بندی داده‌هایی استفاده می‌شود که ویژگی‌های آنها توزیع نرمال دارند. به عنوان مثال، می‌توان از این الگوریتم برای دسته‌بندی متن، تصاویر یا داده‌های عددی استفاده کرد.

پارامتر ها:

- var_smoothing
  - بخشی از بزرگترین واریانس از همه ویژگی ها که برای ثبات محاسبات به واریانس ها اضافه می شود.

Multinomial Naive Bayes

Multinomial Naive Bayes برای دسته‌بندی داده‌هایی استفاده می‌شود که ویژگی‌های آنها توزیع چندجمله‌ای دارند. به عنوان مثال، می‌توان از این الگوریتم برای دسته‌بندی متن یا داده‌های عددی استفاده کرد.

Bernoulli Naive Bayes

Bernoulli Naive Bayes برای دسته‌بندی داده‌هایی استفاده می‌شود که ویژگی‌های آنها می‌توانند فقط دو مقدار داشته باشند، مانند صفر یا یک. به عنوان مثال، می‌توان از این الگوریتم برای دسته‌بندی داده‌های بیتی یا داده‌های خام استفاده کرد.

مقایسه نقاط قوت و ضعف انواع بیز ساده

انواع بیز ساده	Gaussian Naïve Bayes	Multinomial Naïve Bayes	Bernoulli Naïve Bayes
نقاط قوت	ساده و کارآمد برای اجرا. دقت نسبتاً خوب برای داده‌های با توزیع نرمال. می‌تواند برای دسته‌بندی داده‌های عددی، متنی و تصویری استفاده شود.	ساده و کارآمد برای اجرا. دقت نسبتاً خوب برای داده‌های با توزیع چندجمله‌ای. می‌تواند برای دسته‌بندی داده‌های متنی و عددی استفاده شود.	ساده و کارآمد برای اجرا. دقت نسبتاً خوب برای داده‌های دودویی. می‌تواند برای دسته‌بندی داده‌های بیتی و خام استفاده شود.
نقاط ضعف	فرض استقلال ویژگی‌ها ممکن است همیشه صحیح نباشد. ممکن است برای داده‌هایی با توزیع غیر نرمال مناسب نباشد.	فرض استقلال ویژگی‌ها ممکن است همیشه صحیح نباشد. ممکن است برای داده‌هایی با توزیع غیر چندجمله‌ای مناسب نباشد.	فرض استقلال ویژگی‌ها ممکن است همیشه صحیح نباشد. ممکن است برای داده‌هایی با تعداد زیادی ویژگی مناسب نباشد.

مقایسه ویژگی‌های انواع بیز ساده

ویژگی	Gaussian Naïve Bayes	Multinomial Naïve Bayes	Bernoulli Naïve Bayes
نوع توزیع	نرمال	چندجمله‌ای	دودویی
نوع داده	متن، تصویر، عددی	متن، عددی	بیتی، خام
فرض استقلال ویژگی‌ها	بله	بله	بله
دقت	نسبتاً خوب برای داده‌های با توزیع نرمال	نسبتاً خوب برای داده‌های با توزیع چندجمله‌ای	نسبتاً خوب برای داده‌های دودویی

پیاده سازی الگوریتم Navie Bayes

```
# Import necessary libraries
# from sklearn.model_selection import train_test_split
# from sklearn.naive_bayes import GaussianNB
# from sklearn.metrics import accuracy_score


x = df[['sepal_length', 'sepal_width',  'petal_length', 'petal_width']]
y = df['species']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

gnb = GaussianNB()

gnb.fit(x_train, y_train)

y_pred = gnb.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

(K-NN) K Nearest Neighbors

الگوریتم K نزدیک ترین همسایه (K-Nearest Neighbors) که به‌اختصار به آن KNN نیز گفته می‌شود یک الگوریتم یادگیری ماشین با ناظر ساده (Supervised Machine Learning) و پیاده‌سازی آن آسان است. این الگوریتم می‌تواند برای حل مشکلات طبقه‌بندی (Classification) و رگرسیون (Regression) استفاده شود.

KNN هم‌چنین به‌عنوان یک مدل مبتنی بر نمونه (instance-based method) یا یک یادگیرنده‌ی تنبل (lazy learner) شناخته می‌شود؛ زیرا یک مدل داخلی ایجاد نمی‌کند و از داده‌های آموزش عملکرد متمایز را یاد نمی‌گیرد؛ فقط نمونه‌های آموزشی را حفظ می‌کند که به‌عنوان «دانش» برای مرحله‌ی پیش‌بینی استفاده می‌شود.

مراحل الگوریتم K نزدیک‌ترین همسایه

1. داده‌ها را بارگذاری می‌کنیم.
2. مقدار K را تعیین می‌کنیم که همان تعداد نزدیک‌ترین همسایه‌ها هستند.
3. برای هر نمونه داده:
   - فاصله‌ی میان نمونه داده‌ی جدید را با نمونه داده‌های موجود محاسبه می‌کنیم.
   - فاصله و شاخص هر نمونه را به یک فهرست وارد می‌کنیم.

4. کل لیست را براساس فاصله‌ی نمونه داده‌ها، از کمترین به بیشترین فاصله، مرتب می‌کنیم.
5. Kتا از اولین نمونه‌های فهرست مرتب‌شده را به‌عنوان K نزدیک‌ترین همسایه انتخاب می‌کنیم.
6. برچسب این K نمونه را بررسی می‌کنیم.
7. اگر مسئله رگرسیون باشد، میانگین برچسب‌های این K نمونه داده برچسب نمونه داده جدیدمان خواهد بود.
8. درصورتی‌که مسئله طبقه‌بندی باشد، نمونه‌ی جدید هم همان برچسب K همسایه را خواهد داشت.

قطعاً بعد از دیدن این الگوریتم، از اولین سوالاتی که برایمان پیش می‌آید این است که اولاً مقدار K را چگونه انتخاب کنیم و دوماً چطور این نزدیکترین همسایه‌ها را پیدا کنیم. در ادامه پاسخ این دو سؤال را خواهیم یافت.

الگوریتم K-NN چطور کار می کند؟

در این الگوریتم، برای طبقه‌بندی یک نقطه داده جدید، ابتدا K نزدیک‌ترین همسایه به آن نقطه در مجموعه داده‌های آموزشی پیدا می‌شوند. سپس، بر اساس بیشترین تکرار برچسب در میان این K همسایه، برچسب نقطه جدید تعیین می‌شود.

```
# Import necessary libraries
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import accuracy_score


x = df[['sepal_length', 'sepal_width',  'petal_length', 'petal_width']]
y = df['species']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train, y_train)

y_pred = knn.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

چطور مقدار بهینه K یا Optimal K Value را پیدا کنیم؟

```
best_accuracy = 0
best_k = 0

# Loop to check numbers for k from 2 to 20
for k in range(2, 21):
    # Initialize the KNN classifier with the current k value
    knn = KNeighborsClassifier(n_neighbors=k)
```

```
    # Train the classifier
    knn.fit(X_train, y_train)

    # Make predictions
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f'k={k}, Accuracy: {accuracy:.2f}')

    # Update the best accuracy and k if the current accuracy is higher
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

# Print out the best k value
print(f'The best k is {best_k} with an accuracy of {best_accuracy:.2f}')
```

چطور نزدیک‌ترین همسایه‌ها را پیدا کنیم؟

تکنیک‌های مختلفی برای یافتن k نزدیک‌ترین همسایه وجود دارد، می‌توانیم از هر یک از الگوریتم های زیر برای محاسبه فاصله بین نقاط استفاده کنیم.

- فاصله‌ی اقلیدسی (Euclidean distance)
- فاصله‌ی منهتن (Manhattan distance)
- فاصله‌ی مینکوفسکی (Minkowski distance)
- فاصله همینگ (Hamming distance)
- فاصله چبیشف (Chebyshev distance)
- فاصله ماهالانوبیس (Mahalanobis distance)
- فاصله ژاکارد (Jaccard distance)

میتوانیم مقادیر بالا به صورت زیر در کد تنظیم کنیم

```
# Euclidean distance
knn_euclidean = KNeighborsClassifier(n_neighbors=k, metric='euclidean')

# Manhattan distance
knn_manhattan = KNeighborsClassifier(n_neighbors=k, metric='manhattan')

# Minkowski distance
knn_minkowski = KNeighborsClassifier(n_neighbors=k, metric='minkowski',
p=3)  # p can be any positive integer

# Hamming distance
knn_hamming = KNeighborsClassifier(n_neighbors=k, metric='hamming')

# Chebyshev distance
knn_chebyshev = KNeighborsClassifier(n_neighbors=k, metric='chebyshev')

# Mahalanobis distance
```

```
knn_mahalanobis = KNeighborsClassifier(n_neighbors=k,
metric='mahalanobis', metric_params={'V': np.cov(X_train.T)})


# Jaccard distance
knn_jaccard = KNeighborsClassifier(n_neighbors=k, metric='jaccard')
```

پارامتر های K-NN:

- **n_neighbors تعداد همسایگان:** این مقدار عددی تعیین می‌کند که چند همسایه برای پرس و جوهای kneighbors استفاده شود. مقدار پیش‌فرض ۵ است.
- **weights وزن‌ها:**
  - تابع وزن مورد استفاده در پیش بینی مقادیر ممکن:
  - "uniform": وزنهای یکنواخت. همه نقاط در هر محله به طور مساوی وزن داده می شوند.
  - "distance": نقاط وزن برعکس فاصله آنها. در این حالت، همسایگان نزدیکتر یک نقطه پرس و جو تأثیر بیشتری نسبت به همسایگانی که دورتر هستند، خواهند داشت.
  - [callable] : یک تابع تعریف شده توسط کاربر که آرایه ای از فاصله ها را می پذیرد و آرایه ای به همان شکل حاوی وزن ها را برمی گرداند.
- **algorithm الگوریتم:** این پارامتر الگوریتم مورد استفاده برای محاسبه همسایگان نزدیک را مشخص می‌کند. می‌تواند 'ball_tree' ، 'kd_tree'، 'brute'، یا 'auto' تلاش خواهد کرد تا مناسب ترین الگوریتم را بر اساس مقادیر ارسال شده به روش برازش تعیین کند. انتخاب الگوریتم می‌تواند بر سرعت و نیاز به حافظه فرآیند آموزش تأثیر بگذارد.
- **leaf_size اندازه برگ:** این مقدار عددی بر سرعت ساخت و پرس و جو، همچنین حافظه مورد نیاز برای ذخیره درخت هنگام استفاده از 'ball_tree' یا 'kd_tree' تأثیر می‌گذارد. مقدار بهینه بستگی به ماهیت مسئله دارد.
- **P**: این پارامتر قدرت متریک Minkowski است. وقتی p=1 ، این معادل استفاده از فاصله Manhattan است، و برای p=2، فاصله Euclidean است. برای p دلخواه، فاصله Minkowski استفاده می‌شود.
- **Metric متریک:** این پارامتر متریک مورد استفاده برای محاسبه فاصله را تعریف می‌کند. پیش‌فرض 'minkowski'است که نتیجه فاصله استاندارد Euclidean است وقتی p=2 است. همچنین می‌توانید از متریک‌های دیگر مانند'manhattan' ، 'chebyshev'، 'hamming'و غیره استفاده کنید، یا یک تابع قابل تماس که دو آرایه را به عنوان ورودی می‌گیرد و مقدار فاصله را برمی‌گرداند.
- **metric_params پارامترهای متریک:** آرگومان های کلمه کلیدی اضافی برای تابع متریک.
- **n_jobs تعداد کارهای موازی:** این پارامتر تعیین می‌کند که چند کار موازی برای جستجوی همسایگان اجرا شود None .به معنای 1 است مگر اینکه در یک محیط joblib.parallel_backend باشد. -1 به معنای استفاده از تمام پردازنده‌ها است.

Decision Tree

درخت های تصمیم (DTs) روش های بدون پارامتر از یادگیری با نظارت هستند که برای دسته بندی و رگرسیون استفاده می شوند. هدفت پیاده سازی مدلی است که ارزش متغیر هدف را با یادگیری قوانین تصمیم گیری ساده استنتاج شده از ویژگی های داده، پیش بینی کند.

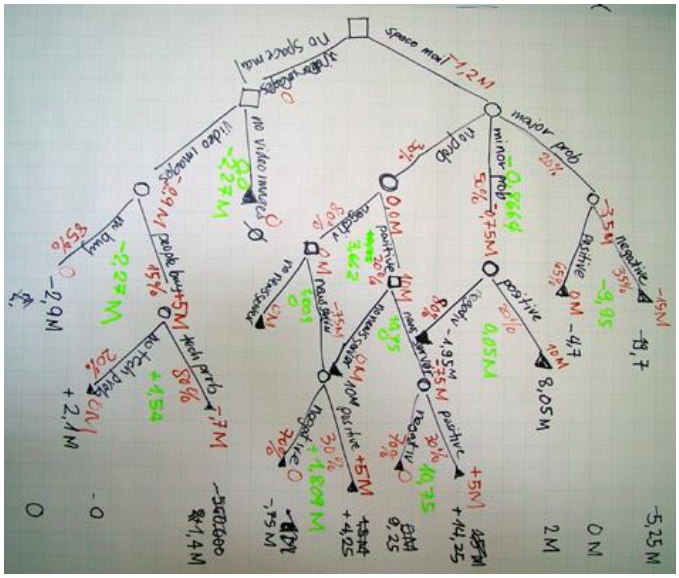
برخی مزایای استفاده از درخت تصمیم

1. درک و تفسیر آن ساده است.
2. درختان را می توان تجسم کرد.
3. نیازی نیست داده های خودمونو به عدد تبدیل کنیم.

4. نیاز به آماده سازی داده های کمی دارد. سایر تکنیک ها اغلب به نرمال سازی داده ها نیاز دارند، متغیرهای ساختگی باید ایجاد شوند و مقادیر خالی حذف شوند.
5. اردر زمانی یا هزینه استفاده از الگوریتم درخت تصمیم برای train کردن درخت لوگاریتمی است.
6. توانایی هندل کردن مسائل با چند دسته بندی.
7. امکان اعتبارسنجی مدل با استفاده از آزمون های آماری. این امر باعث می شود که قابلیت اطمینان مدل در نظر گرفته شود.

یک توضیح ساده

در بازی بیست سؤالی، بازیکن باید یک درخت تصمیم در ذهن خود بسازد که به خوبی موارد را از هم جدا کند تا با کمترین سؤال به جواب برسد. در صورتی بازیکن به جواب می‌رسد که درخت ساخته شده بتواند به خوبی موارد را از هم جدا کند، الگوریتم درخت تصمیم هم به همین شیوه کار می کند.



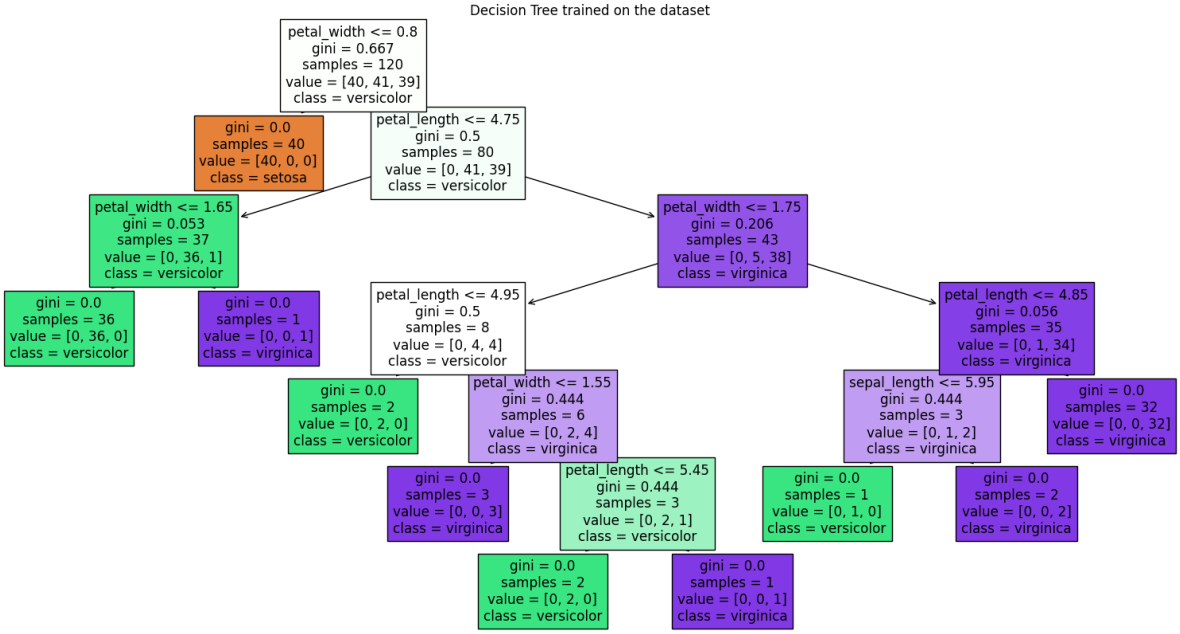
یک مثال:

```
# Import necessary libraries
# from sklearn import tree
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import accuracy_score


x = df[['sepal_length', 'sepal_width',  'petal_length', 'petal_width']]
y = df['species']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

dt = tree.DecisionTreeClassifier()

dt.fit(x_train, y_train)

y_pred = dt.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

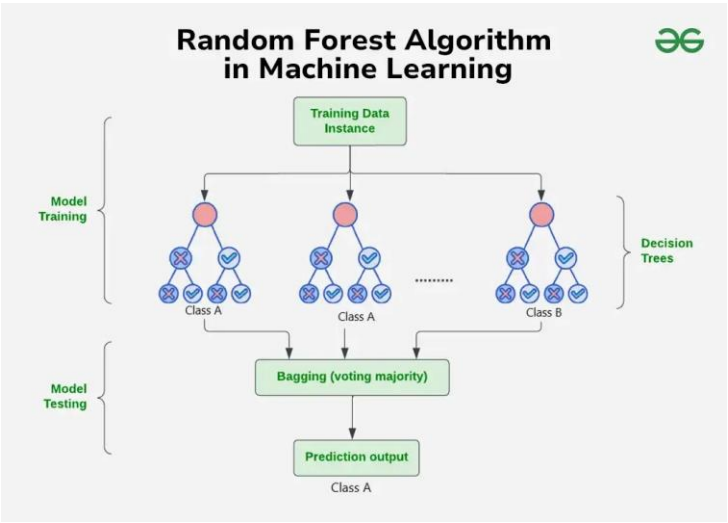
میتونیم به روش زیر درخت رو به نمایش در بیاریم.

```
plt.figure(figsize=(20, 10))
tree.plot_tree(dt, filled=True, feature_names=x.columns,
class_names=y.unique())
plt.title('Decision Tree trained on the dataset')
plt.show()
```



Random Forest

الگوریتم جنگل تصادفی یک تکنیک یادگیری ماشین بسیار قوی است. این الگوریتم با ساخت تعداد زیادی درخت تصمیم در فاز آموزش کار می‌کند. هر درخت با زیردیتا یا زیر مجموعه ای تصادفی از دیتاست اصلی آموزش می‌بیندکه این امر باعث کاهش خطا و overfiting می‌شود. در هنگام پیشبینی هر درخت با توجه به ویژگی های ورودی یک پیشبینی ای را انجام میدهد که تمام این پیشبینی ها جمع آوری می‌شوند که امری شبیه به رای گیری است در نهایت خروجی ای که بیشترین رای را دارد به عنوان خروجی اصلی انتخاب می شود.



نمونه

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

x = df[['sepal_length', 'sepal_width',  'petal_length', 'petal_width']]
y = df['species']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

model = RandomForestClassifier(n_estimators = 100)
model.fit(x_train, y_train)

prediction = model.predict(x_test)

print("accuracy: ", accuracy_score(y_test, prediction))
print(classification_report(y_test, prediction))
```

پارامتر های الگوریتم جنگل تصادفی

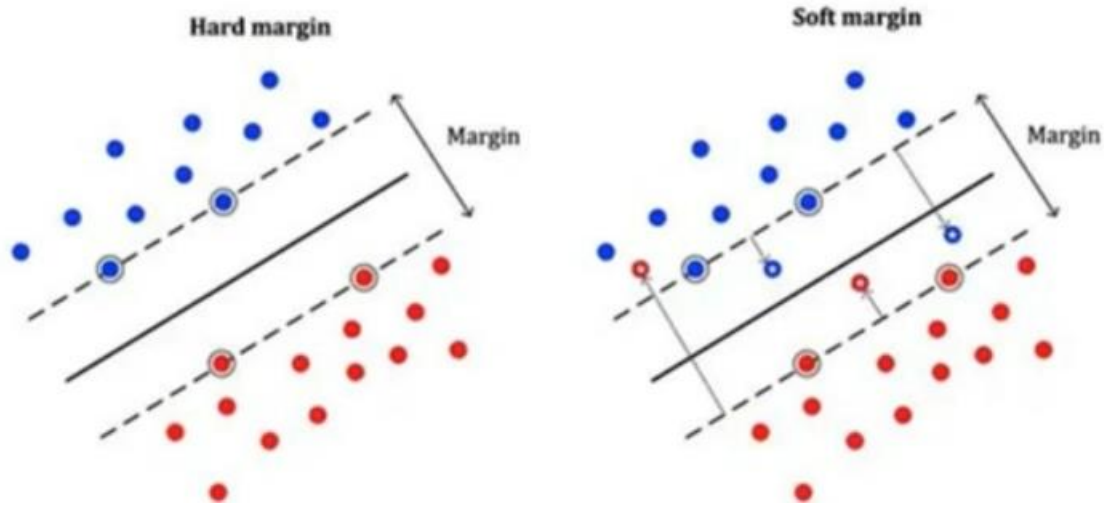
- n_estimators: تعداد درخت های تصمیم.
- max_depth: حداکثر عمق مجاز برای هر درخت تصمیم را در جنگل تصادفی مشخص می‌کند. درخت عمیق‌تر می‌تواند روابط پیچیده‌تری را در داده‌ها ثبت کند، اما اگر به‌درستی کنترل نشود، ممکن است منجر به بیش برازش شود.
- max_features: تعداد ویژگی هایی که هنگام جستجوی بهترین تقسیم باید در نظر بگیرید.

SVM(Support Vector Machine)

ماشین بردار پشتیبان یا Support Vector Machine که به‌اختصار به آن SVM گفته می‌شود یک الگوریتم یادگیری ماشین با ناظر است که نمونه‌ی داده‌ها را با استفاده از یک خط یا هایپرپلین (Hyperplane)، از هم جدا می‌کند. این جداسازی به‌گونه‌ای است که نقاط داده‌ای که در یک طرف خط هستند مشابه‌به‌هم و در یک گروه قرار می‌گیرند. نمونه‌داده‌های جدید هم بعد از اضافه‌شدن به همان فضا در یکی از دسته‌های موجود قرار خواهند گرفت.

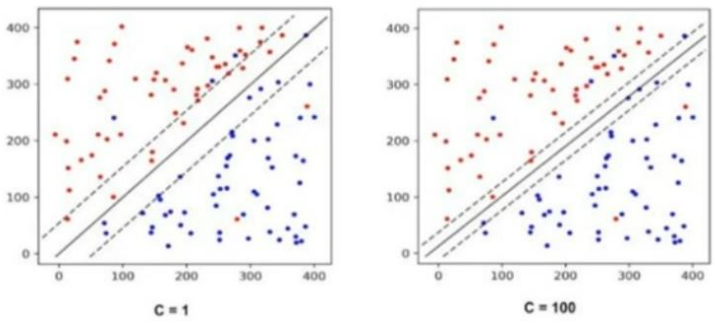
برای درک نحوه‌ی عملکرد ماشین بردار پشتیبان (SVM) بیایید مثال خرگوش و ببر را در نظر بگیریم. بیایید اکنون یک سناریوی کوچک را در نظر بگیریم و وانمود کنیم صاحب مزرعه‌ای هستیم و بنا به دلایلی می‌خواهیم حصاری برای محافظت از خرگوش‌های خود دربرابر ببرها ایجاد کنیم.

اصل اساسی ماشین بردار پشتیبان این است که یک هایپرپلین ترسیم کنیم که دو کلاس را به‌بهترین شکل از یکدیگر جدا کند. حال درمورد مثال ما دو کلاس خرگوش و ببر هستند؛ بنابراین ما با ترسیم یک هایپرپلین رندوم شروع می‌کنیم و سپس فاصله‌ی میان هایپرپلین و نزدیک‌ترین نقاط داده‌ی هر کلاس را بررسی می‌کنیم.

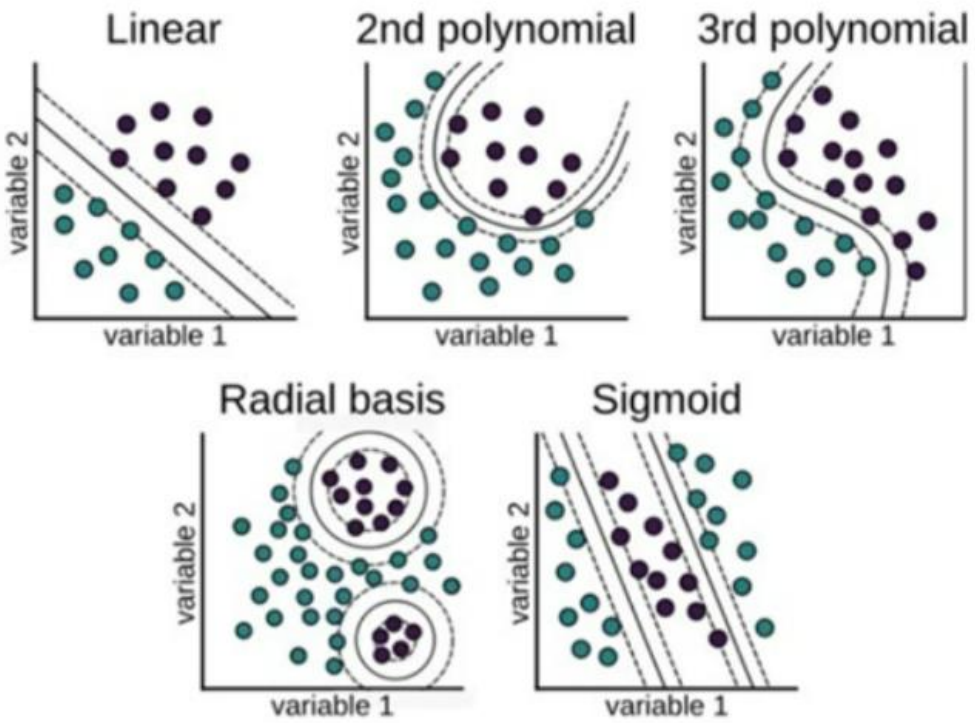


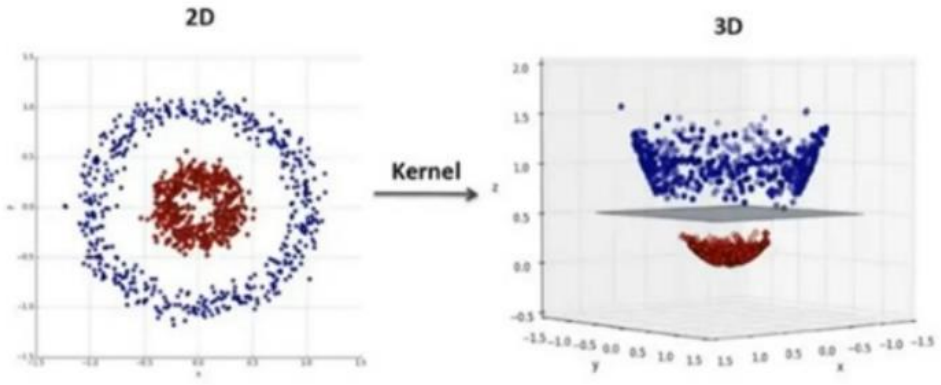
پارامتر های SVM:

- C: این پارامتر در الگوریتم ماشین بردار پشتیبان (SVM) نقش بسیار مهمی ایفا می‌کند. این پارامتر تعادل بین "مرزها" یا حاشیه‌ای که بین دسته‌ها ایجاد می‌شود و تعداد نقاطی که اشتباهاً دسته‌بندی شده‌اند را تنظیم می‌کند، به عبارت دیگر، پارامتر C در SVM میزان خطای قابل قبول را تعیین می‌کند. اگر C بزرگ باشد، الگوریتم سعی می‌کند خطای طبقه‌بندی را کاهش دهد، حتی در صورتی که حاشیه کوچکتر شود. اما اگر C کوچک باشد، الگوریتم می‌پذیرد که برخی از نقاط به اشتباه طبقه‌بندی شوند تا حاشیه بزرگتری داشته باشد، به طور کلی، پارامتر C در SVM یک پارامتر تنظیمی است که تعادل بین پیچیدگی مدل (یعنی حاشیه) و خطای آموزش (یعنی نقاط اشتباهاً دسته‌بندی شده) را کنترل می‌کند.



- Kernel: هنگامی‌که نقاط داده را نمی‌توان با یک خط مستقیم یا یک هایپرپلین مستقیم جدا کرد، مسئله غیرخطی نامیده می‌شود. در چنین شرایطی کرنل‌های ماشین بردار پشتیبان (SVM) وارد عمل می‌شوند و ابعاد فضا را افزایش می‌دهند تا نقاط داده به‌صورت خطی تفکیک‌پذیر شوند.
  - مقادیر: ***{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'***





- Degree: "درجه" معمولاً به کرنل چندجمله‌ای اشاره دارد. کرنل چندجمله‌ای یکی از روش‌هایی است که برای تبدیل فضای ویژگی‌ها به فضایی با بعد بالاتر استفاده می‌شود. این تبدیل به SVM اجازه می‌دهد تا داده‌هایی که در فضای اصلی خطی قابل جداسازی نیستند را در فضای با بعد بالاتر جدا کند. درجه کرنل چندجمله‌ای تعیین می‌کند که چه میزان تعامل بین ویژگی‌ها در نظر گرفته شود. به عنوان مثال، اگر درجه برابر با 2 باشد، تنها تعامل بین دو ویژگی در نظر گرفته می‌شود. به طور کلی، انتخاب درجه مناسب برای کرنل چندجمله‌ای یک مسئله تنظیم مدل است و می‌توان با استفاده از روش‌هایی مانند Grid SearchCV بهترین مقدار را پیدا کرد.

مثال:

```
from sklearn.svm import SVC


x = df[['sepal_length', 'sepal_width',  'petal_length', 'petal_width']]
y = df['species']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

model = SVC()
model.fit(x_train, y_train)

prediction = model.predict(x_test)

print("accuracy: ", accuracy_score(y_test, prediction))
```

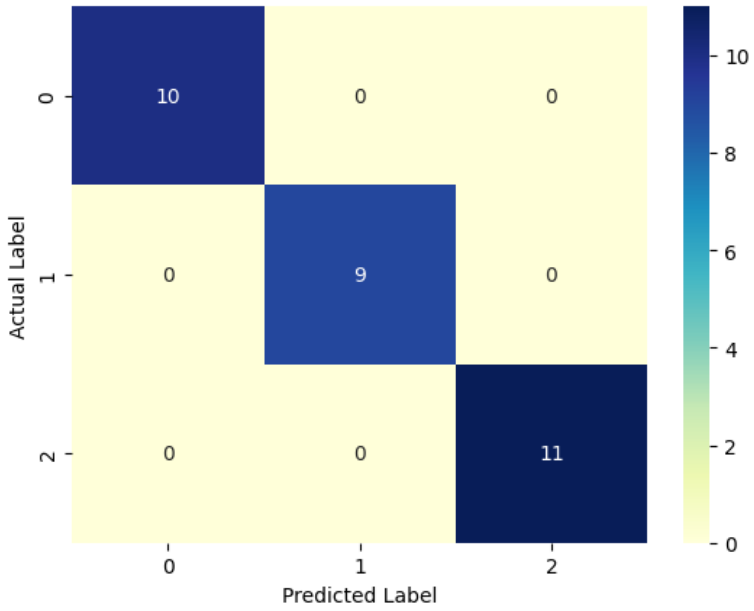
ماتریس درهم ریختگی (Confusion Matrix)

پس از اجرای الگوریتم دسته‌بندی، با توجه به توضیحات و تعاریف ذکر شده، می‌توان عملکرد یک طبقه‌بند را به کمک جدولی به شکل زیر بررسی کرد. این جدول تعداد پیشبینی های درست و غلط را نمایش میدهد.

		برچسب پیش‌بینی شده	
		مثبت	منفی
برچسب شناخته شده	مثبت	TP	FN
	منفی	FP	TN

یک نمونه از این جدول را مشاهده کنید.

```
from sklearn.metrics import confusion_matrix

conf_m = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_m, annot=True, cmap = "YlGnBu")
plt.ylabel("Actual Label")
plt.xlabel("Predicted Label")
```



Classification Report

تابع classification_report در کتابخانه scikit-learn یک گزارش متنی ایجاد می‌کند که معیارهای اصلی دسته‌بندی را نشان می‌دهد. این تابع معیارهایی مانند دقت (precision)، بازیابی (recall)، و امتیاز F1 (F1 score) را برای هر کلاس محاسبه می‌کند و همچنین پشتیبانی (support)، که تعداد نمونه‌های واقعی برای هر کلاس است، را گزارش می‌دهد.

مشاهده یک مثال:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

در گزارش دسته‌بندی (classification_report)، دقت (precision) و بازیابی (recall) دو معیار ارزیابی عملکرد مدل‌های دسته‌بندی هستند:

- دقت :(Precision)این معیار نشان می‌دهد که از بین تمام نمونه‌هایی که مدل به عنوان مثبت پیش‌بینی کرده است، چند درصد واقعاً مثبت هستند. به عبارت دیگر، دقت نشان‌دهنده کیفیت پیش‌بینی‌های مثبت مدل است. فرمول محاسبه دقت به صورت زیر است:

$$Precision = \frac{TP}{TP + FP}$$

که در آن (TP) تعداد مثبت‌های واقعی (True Positives) و (FP) تعداد مثبت‌های کاذب (False Positives) است.

- بازیابی :(Recall)این معیار نشان می‌دهد که از بین تمام نمونه‌های واقعاً مثبت، مدل چند درصد را به درستی به عنوان مثبت شناسایی کرده است. بازیابی نشان‌دهنده توانایی مدل در پیدا کردن تمام نمونه‌های مثبت است. فرمول محاسبه بازیابی به صورت زیر است:

$$Recall = \frac{TP}{TP + FN}$$

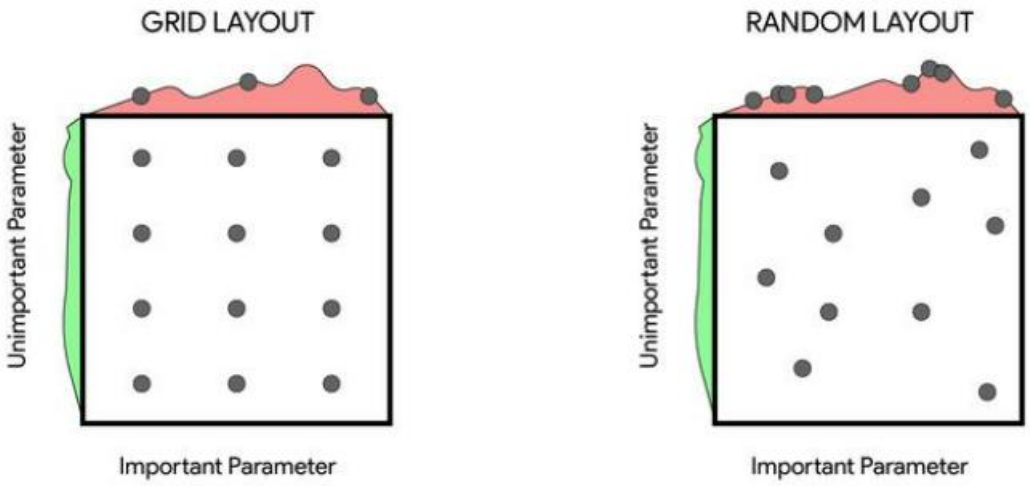
که در آن (TP) همان تعداد مثبت‌های واقعی است و (FN) تعداد منفی‌های کاذب (False Negatives) است. این دو معیار به ما کمک می‌کنند تا درک بهتری از عملکرد مدل در دسته‌بندی داده‌ها داشته باشیم، به خصوص وقتی که توزیع کلاس‌ها متوازن نیست یا هزینه‌های خطاهای مختلف متفاوت است.

بهینه سازی (Hyperparameter Optimization)

Hypertparameter ها پارامترهایی هستند که ما برای آموزش مدل تنظیم می‌کنیم. این فراپارامتر ها تاثیرات عمده ای بر دقت و کارایی مدل در حین آموزش مدل دارند. از این رو نیاز است که با دقت و به درستی تنظیم شوند تا به نتایج بهتر و کارآمد تری دست پیدا کنیم.

دو روش اصلی برای این کار وجود دارد

- Grid Search:
  - در Grid Search تمام مقادیر ممکن هایپرپارامترها در مجموعه تعریف شده است. سپس این مجموعه از مقادیر ممکن ابرپارامترها با استفاده از ضرب دکارتی ترکیب شده و یک شبکه چند بعدی را تشکیل می دهند.
- Random Search: این یکی دیگر از انواع جستجوی گرید است که در آن به جای اینکه تمام نقاط شبکه را امتحان کنیم، نقاط تصادفی را امتحان می کنیم.



مثال 1:

فرض رو بر این میزاریم که یک مدل Naïve bayes رو آموزش دادیم. حالا باید بهرین پارامتر هارو برای اون پیدا کنیم تا دقت مدل رو افزایش بدیم:

```
from sklearn.model_selection import GridSearchCV

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=gnb,
                 param_grid=params_NB,
                #  cv=cv_method,    # use any cross validation technique
                 verbose=1,
                 scoring='accuracy')
gs_NB.fit(x_train, y_train)

gs_NB.best_params_
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
{'var_smoothing': 0.02310129700083159}
```

این کد یک فرآیند بهینه‌سازی برای پیدا کردن بهترین پارامترها برای مدل Naive Bayes را انجام می‌دهد. این کد از الگوریتم Grid Search استفاده می‌کند تا مقادیر مختلفی از پارامتر `var_smoothing` را امتحان کند و بهترین مقدار برای آن را پیدا کند. این پارامتر مربوط به اسموثینگ (smoothing) در مدل Naive Bayes است که برای جلوگیری از احتمال صفر شدن در مواقعی که داده‌ها در دسته‌های مختلف نیستند، استفاده می‌شود.

مثال 2:

حالا همین کارو برای مدل knn خودمون انجام می دیم:

```
from sklearn.model_selection import GridSearchCV

# Define the parameters to test
parameters = {
    'n_neighbors' : range(1, 30),
    'weights': ['uniform', 'distance']
}

# Create grid search instance
knn_grid_search = GridSearchCV(estimator = knn,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           return_train_score = True)
knn_grid_search.fit(x_train, y_train)

knn_grid_search.best_params_
```

```
{'n_neighbors': 3, 'weights': 'uniform'}
```

برای SVM:

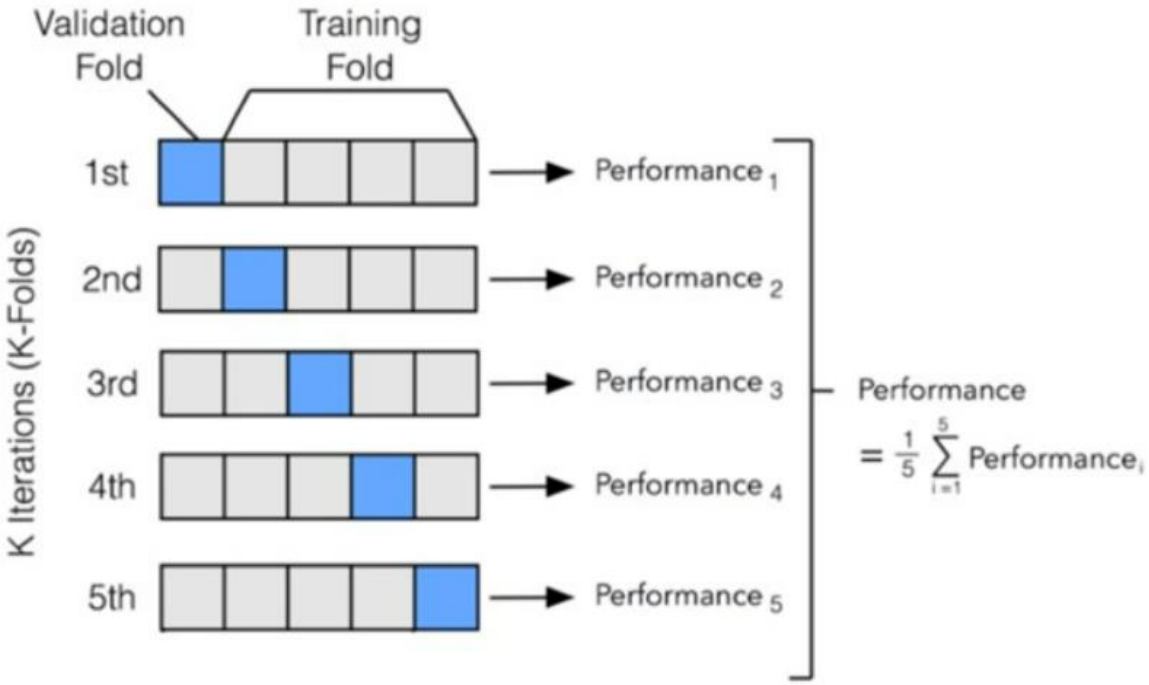
```
from sklearn.model_selection import GridSearchCV, KFold

param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [1, 0.1, 0.01],
    'kernel': ['poly', 'rbf'],
```

```
    'degree': range(1, 5)
}

SVC_model_tuned = GridSearchCV(estimator=SVC_model, param_grid=param_grid)
SVC_model_tuned.fit(x_train, y_train)

SVC_model_tuned.best_params_
```

```
{'C': 0.1, 'degree': 3, 'gamma': 0.1, 'kernel': 'poly'}
```

K-fold Cross Validation

اعتبارسنجی متقابل یا cross validation یک روش برای ارزیابی و بررسی تعمیم‌پذیری (Generalization) مدل است. زمانی که مدل بتواند در مورد داده‌هایی که قبلا ندیده، پیش‌بینی دقیقی داشته باشد، یعنی تعمیم‌پذیری بالایی دارد.به داده‌هایی که مدل در فرآیند آموزش آن‌ها را دیده، seen data و به داده‌هایی که قبلا ندیده، unseen data گفته می‌شود. در k fold cross validation، دیتاست به k شکل مختلف تقسیم می‌شود که به هرکدام یک fold می‌گوییم. مدل k بار روی این k fold آموزش داده می‌شود. به این صورت k دقت به دست آمده و در نهایت میانگین این دقت‌ها محاسبه می‌شود.



بررسی دقت مدل knn:

```
from sklearn.model_selection import cross_val_score

cvs = cross_val_score(estimator = knn, X = x_train, y = y_train, cv = 10)
cvs.mean()
```

```
0.95
```

بهبود در بهینه سازی با K fold:

```
from sklearn.model_selection import KFold, GridSearchCV

# Define the hyperparameters to search over
param_grid = {
    'n_neighbors': range(1, 30),  # Example values, adjust as needed
    'weights': ['uniform', 'distance'],
}

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=kf)
grid_search.fit(x_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

print("Best parameters:", best_params)

Best parameters: {'n_neighbors': 12, 'weights': 'uniform'}
```

حالا می‌بینید بر خلاف بالا که نوشته بود تعداد همسایه ها 3 تا باشه الان نوشته 12 تا بهترین حالت است.

یادگیری بدون ناظر

خوشه بندی (Clustering)

در تجزیه و تحلیل خوشه یا خوشه‌بندی، گروه‌بندی مجموعه‌ای از اشیاء انجام می‌شود، اینکار به این صورت است که اشیاء در یک گروه (به نام خوشه) در مقایسه با دیگر دسته‌ها (خوشه‌ها) مشابه‌تر هستند. این وظیفهٔ اصلی داده‌کاوی اکتشافی است و یک روش معمول برای تجزیه و تحلیل داده‌های آماری است.

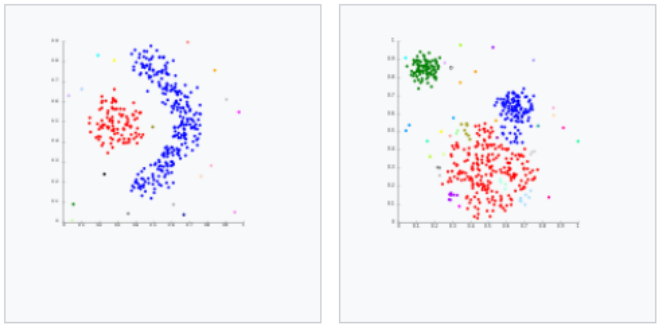
احتمالاً بیش از ۱۰۰ الگوریتم خوشه‌بندی منتشر شده وجود دارد. همه مدل‌ها برای خوشه‌هایشان بیان نشده‌اند، بنابراین نمی‌توان به راحتی دسته‌بندی کرد.

انواع الگوریتم ها:

1. **خوشه‌بندی براساس اتصال (خوشه‌بندی سلسله‌مراتبی)**
   a. **خوشه‌بندی تک‌پیوندی (Single-Linkage Clustering):** این روش که به روش Bottom-Up و Agglomerative نیز معروف است روشی است که در آن ابتدا هر داده به عنوان یک خوشه در نظر گرفته می‌شود. در ادامه با به‌کارگیری یک الگوریتم هر بار خوشه‌های دارای ویژگی‌های نزدیک به هم با یکدیگر ادغام شده و این کار ادامه می‌یابد تا به چند خوشهٔ مجزا برسیم. مشکل این روش حساس بودن به نویز و مصرف زیاد حافظه می‌باشد.
   b. **خوشه‌بندی کامل پیوند (Complete-Linkage Clustering):** در این روش که به روش Top-Down و Divisive نیز معروف است ابتدا تمام داده‌ها به عنوان یک خوشه در نظر گرفته شده و با به‌کارگیری

یک الگوریتم تکرار شونده هربار داده‌ای که کمترین شباهت را با داده‌های دیگر دارد به خوشه‌های مجزا تقسیم می‌شود. این کار ادامه می‌یابد تا یک یا چند خوشه یک عضوی ایجاد شود. مشکل نویز در این روش برطرف شده‌است.

مثال هایی از خوشه linkage



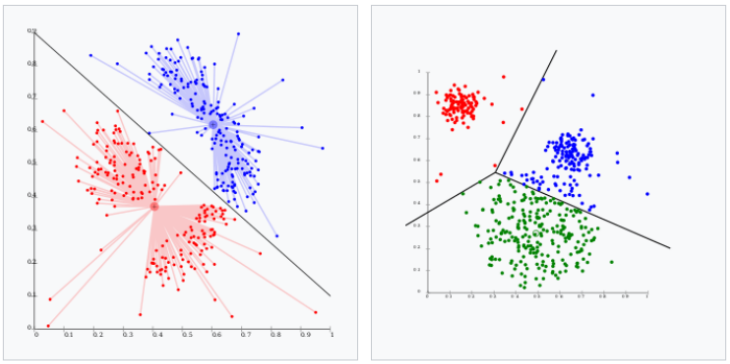
single linkage بر روی داده‌های گوسی

single linkage بر روی خوشه براساس density

2. **خوشه بندی براساس مرکزوار (centroid)**

a. در خوشه بندی براساس مرکزوار، خوشه‌ها با یک بردار مرکزی نشان داده می‌شوند، که ممکن است لزوماً جزء مجموعه داده نباشد. هنگامی که تعدادی از خوشه‌ها به k متصل می‌شوند، خوشه بندی k-means یک تعریف رسمی را به عنوان یک مسئله بهینه‌سازی ارائه می‌دهد.

مثال هایی از خوشه بندی k-means



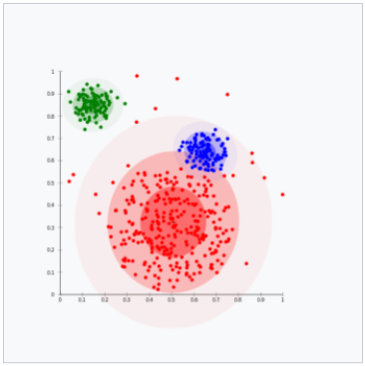
جداسازی داده‌های K-means در Voronoi-cells

عدم نمایش K-means برای خوشه‌های براساسdensity

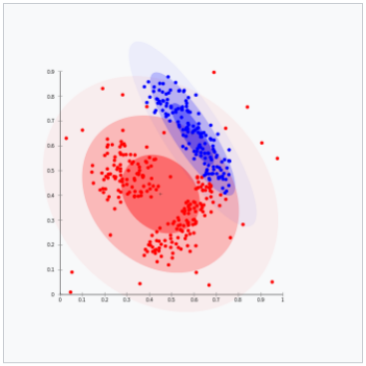
3. **خوشه بندی براساس توزیع**

a. این مدل خوشه بندی که دقیقاً مربوط به آمار می‌باشد، بر اساس مدل‌های توزیع است. خوشه‌ها به راحتی می‌توانند به عنوان اشیایی تعریف می‌شوند که به احتمال زیاد ب توزیع یکسانی دارند. یک ویژگی خوب این رویکرد این است که با نمونه برداری از اشیاء تصادفی از یک توزیع، دقیقاً شبیه نحوه تولید مجموعه داده‌های مصنوعی است. مبنای نظری این روش‌ها عالی است، ولی مشکل اصلی overfitting دارند، مگر اینکه محدودیت‌ها بر پیچیدگی مدل قرار بگیرد. یک روش شناخته شده، مدل مخلوط گاوس (با استفاده از الگوریتم حداکثر سازی انتظار) است. مجموعه داده‌ها معمولاً با یک ثابت (برای جلوگیری از overfitting) تعداد توزیع‌های گاوسی که به صورت تصادفی استفاده شده و به منظور مناسب تر کردن مجموعه داده مدل، پارامترهای آن به‌طور تکراری بهینه شده‌است که به یک بهینه محلی همگرا می‌شود، بنابراین در طول چند اجرا ممکن است نتایج متفاوتی تولید کند.

مثال های Expectation-maximization (EM)



برای داده‌های گوسی، em به خوبی عمل کرده‌است.

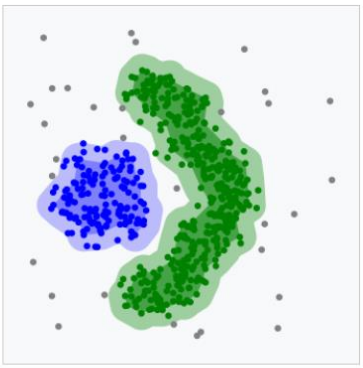


خوشه بندی براساس Density توزیع گوسی را به خوبی مدل نمی‌کند.

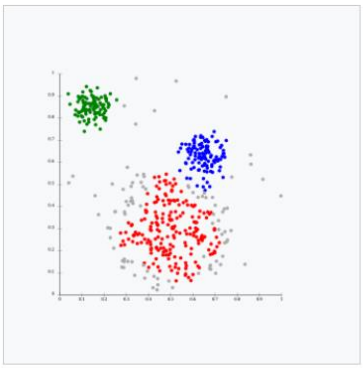
4. خوشه بندی براساس Density

a. در این تکنیک این اصل مطرح می‌شود که خوشه‌ها مناطقی با چگالی بیشتر هستند که توسط مناطق با چگالی کمتر از هم جدا شده‌اند. یکی از مهم‌ترین الگوریتم‌ها در این زمینه الگوریتم DBSCAN است. روش این الگوریتم به این صورت است که هر داده متعلق به یک خوشه در دسترس چگالی سایر داده‌های همان خوشه است، ولی در دسترسی چگالی سایر داده‌های خوشه‌های دیگر نیست. (چگالی داده همسایگی به مرکز داده و شعاع همسایگی دلخواه ε است) مزیت این روش این است که تعداد خوشه‌ها به صورت خودکار مشخص می‌شود. در تشخیص نویز نیز بسیار کاراست.

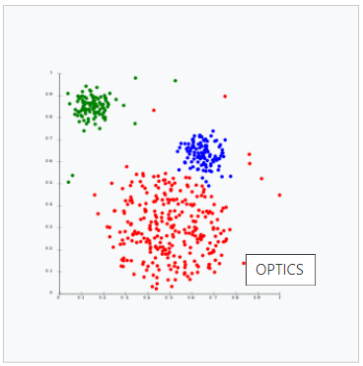
مثال هایی برای خوشه بندی براساس Density



خوشه بندی براساس Density با DBSCAN



DBSCAN و خوشه‌هایی باdensity مشابه .



OPTICS

Hopkins test

آزمون هاپکینز، تصادفی بودن فضایی متغیرها را مورد بررسی قرار می‌دهد. فرض صفر این آزمون می‌گوید توزیع داده‌ها غیرتصادفی و یکنواخت است.

هرچه نتیجه تست هاپکینز به 1 نزدیکتر باشد به این معنی است که توزیع داده‌ها تصادفی است که دلالت بر وجود خوشه‌ها دارد.

یک نمونه کد برای انجام اینکار

```
from sklearn.neighbors import NearestNeighbors
from random import sample
from numpy.random import uniform
import numpy as np
from math import isnan

# Defining the Hopkins test function
def hopkins(X):
    d = X.shape[1]
    #d = len(vars) # columns
    n = len(X) # rows
    m = int(0.1 * n) # heuristic from article [1]
    nbrs = NearestNeighbors(n_neighbors=1).fit(X.values)

    rand_X = sample(range(0, n, 1), m)

    ujd = []  # distances from random points to their nearest neighbors
    wjd = []  # distances from uniformly distributed points to their
nearest neighbors

    for j in range(0, m):
        u_dist, _ =
nbrs.kneighbors(uniform(np.amin(X,axis=0),np.amax(X,axis=0),d).reshape(1,
-1), 2, return_distance=True)
        ujd.append(u_dist[0][1])
        w_dist, _ = nbrs.kneighbors(X.iloc[rand_X[j]].values.reshape(1, -
1), 2, return_distance=True)
        wjd.append(w_dist[0][1])

    H = sum(ujd) / (sum(ujd) + sum(wjd))
    if isnan(H):
        print (ujd, wjd)
        H = 0

    return H

hopkins(df)
```

```
# Applying Hopkins test function on the imported data set
hopkins(veg_df)
```

```
0.9607421962080679
```

As shown above, the hypothesis test value is 0.96 which is quite close to 1, so we can reject the null hypothesis and this means that the data distribution is random implying the presence of clusters.

Source: https://github.com/mhatim99/Hopkins-Test/blob/master/Hopkins%20Test.ipynb

یک نمونه کد دیگر

```
import numpy as np
from sklearn.neighbors import NearestNeighbors

def hopkins_statistic(X, m=0.1):
    """
    Calculate the Hopkins statistic for cluster tendency assessment.

    Parameters:
        X (numpy.ndarray): Input data with shape (n, m).
        m (float): Proportion of samples to use for comparison (default is
0.1).

    Returns:
        float: Hopkins statistic value.
    """
    n, d = X.shape
    rand_X = np.random.choice(range(n), size=int(m * n), replace=False)

    # Fit nearest neighbors model
    nbrs = NearestNeighbors(n_neighbors=2, algorithm='brute').fit(X)

    ujd = []  # distances from random points to their nearest neighbors
    wjd = []  # distances from uniformly distributed points to their
nearest neighbors

    for j in range(len(rand_X)):
        u_dist, _ = nbrs.kneighbors(np.random.normal(size=(1,
d)).reshape(1, -1), 2, return_distance=True)
        ujd.append(u_dist[0][1])

        w_dist, _ = nbrs.kneighbors(X[rand_X[j]].reshape(1, -1), 2,
return_distance=True)
        wjd.append(w_dist[0][1])

    H = sum(ujd) / (sum(ujd) + sum(wjd))
    return H


hopkins_value = hopkins_statistic(scale(df))
print(f"Hopkins statistic value: {hopkins_value:.4f}")
```

الگوریتم k-means

- K در k-means تعداد خوشه ها را مشخص می کند، پس باید آن را به عنوان یک پارامتر در نظر بگیریم.
- مقادیر داخل هر خوشه باید شبیه به یکدیگر باشند.
- مقادیر داخل خوشه ها نباید شبیه به خوشه های دیگر باشند.

مراحل

- ما باید مرکز خوشه را تعیین کنیم.
- ما باید مثال هارا خارج از مرکز با توجه به فاصله دسته بندی کنیم.
- این قدم هارا برای چندین بار تکرار میکنیم تا به حالت پیادار برسیم.

روش های تعیین تعداد بهینه خوشه ها به دو دسته کلی روش های مستقیم یا Direct Methods و روش های مبتنی بر آزمون های آماری یا Statistical Testing Methods تقسیم می‌شوند:

روش های مستقیم تعیین تعداد بهینه خوشه ها در الگوریتم های خوشه بندی: این روش ها به دنبال بهینه سازی یک معیار به‌خصوص، مانند مجموع مربعات فواصل درون خوشه ای (Within-cluster Sum of Square یا WSS) یا سیلوئت میانگین (Average Silhouette) هستند. از جمله این متدها می‌توان به متد elbow و روش های مبتنی بر معیار silhouette اشاره کرد.

روش های مبتنی بر آزمون های آماری در تعیین تعداد بهینه خوشه ها در الگوریتم های خوشه بندی: این متدها به دنبال تطبیق مشاهدات با فرض صفر یک آزمون آماری هستند. از جمله این روش ها می‌توان به Gap Statistics اشاره کرد. علاوه بر متدهای elbow، silhouette و gap statistics، بیش از 30 مورد روش و شاخص مختلف برای تعیین تعداد بهینه خوشه ها در مقالات مختلف ارائه شده است.

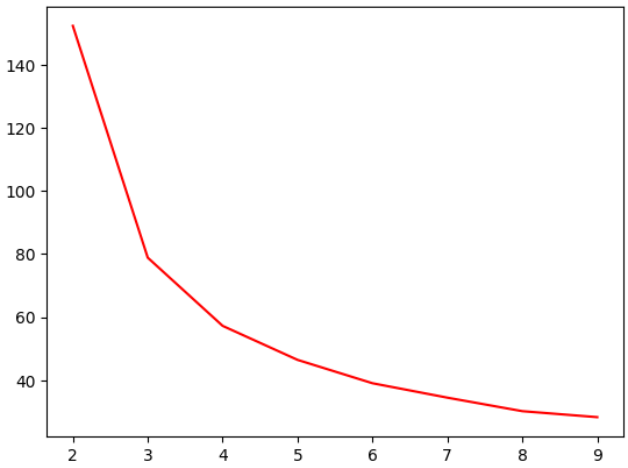
The Elbow Method

روش Elbow یک روش برای تعیین تعداد بهینه‌ی خوشه‌ها در الگوریتم k-means است. این روش به شکل زیر عمل می‌کند:

1. ابتدا k-means را با تعداد خوشه‌های مختلف (مثلاً از ۱ تا n) اجرا می‌کنیم.
2. برای هر تعداد خوشه، میانگین مربوط به فاصله‌ی هر نقطه از مرکز خوشه‌ای که به آن تعلق دارد، محاسبه می‌کنیم.
3. سپس مجموع این میانگین‌ها را برای همه‌ی تعداد خوشه‌ها محاسبه می‌کنیم.
4. نقطه‌ای که در این مرحله به دست می‌آید، نقطه‌ی بهینه‌ی تعداد خوشه‌ها است. به عبارت دیگر، این نقطه نقطه‌ی "البو" است که در نمودار میانگین فاصله‌ها به تعداد خوشه‌ها رسم می‌شود.

```
from sklearn.cluster import KMeans

rate = []

k = range(2, 10)

for i in k:
   k_means = KMeans(n_clusters = i, random_state = 42)
   k_means.fit(df.iloc[:, 0:4])
```

```
    rate.append(k_means.inertia_)

plt.plot(k, rate, "r")
```



همانطور که در تصویر مشخص است شکستگی در نقطه 3 وجود دارد، پس بهترین تعداد برای کلاستر ها عدد 3 است.

در مدل‌سازی خوشه‌بندی K-Means، مقدار inertia یک معیار مهم است. این مقدار نشان‌دهنده این است که داده‌ها به چه اندازه توسط الگوریتم K-Means خوشه‌بندی شده‌اند. برای محاسبه inertia، فاصله بین هر نقطه داده و مرکز خوشه‌ای که به آن تعلق دارد، محاسبه می‌شود. سپس این فاصله را مربع می‌کنیم و مجموع مربع‌ها را در یک خوشه محاسبه می‌کنیم. یک مدل خوب، مدلی است که دارای inertia کم و تعداد خوشه‌ها (K) نیز کم باشد. به عبارت دیگر، مدلی که خوشه‌بندی خوبی انجام داده باشد و تعداد خوشه‌ها به حداقل ممکن رسیده باشد، مدل مطلوبی است.

حالا یک نمونه خوشه بندی انجام بدیم

```
from sklearn.cluster import KMeans

k_means = KMeans(n_clusters=3, random_state=42)
k_means.fit(df.iloc[:, 0:4])

clusters = k_means.labels_
clusters
```

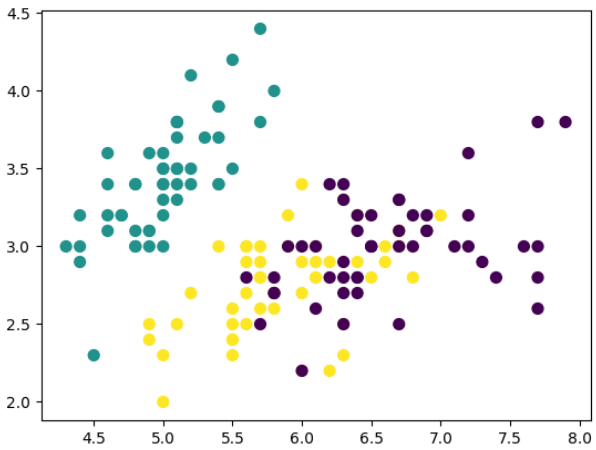
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      dtype=int32)
```

پیشبینی

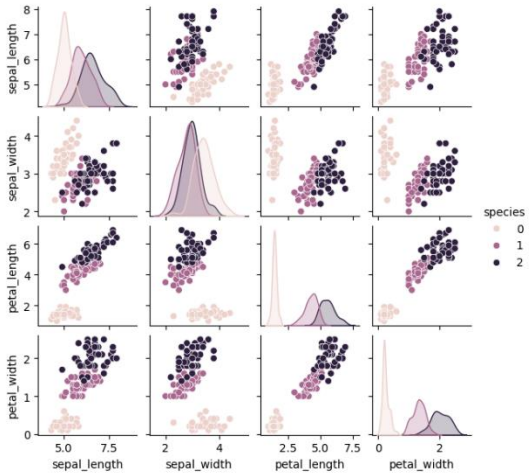
```
k_means.predict([[4.8,  4.0,  4.4,  4.1]])
```

نمایش نمودار با دوتا از ستون ها

```
plt.scatter(df.iloc[:,0], df.iloc[:,1], c= clusters, s = 50)
```



```
sns.pairplot(df, hue='species', height= 1.5)
```



حالا میخوایم ببینیم کلاسترمون چقدر نزدیک به لیبل های واقعیه

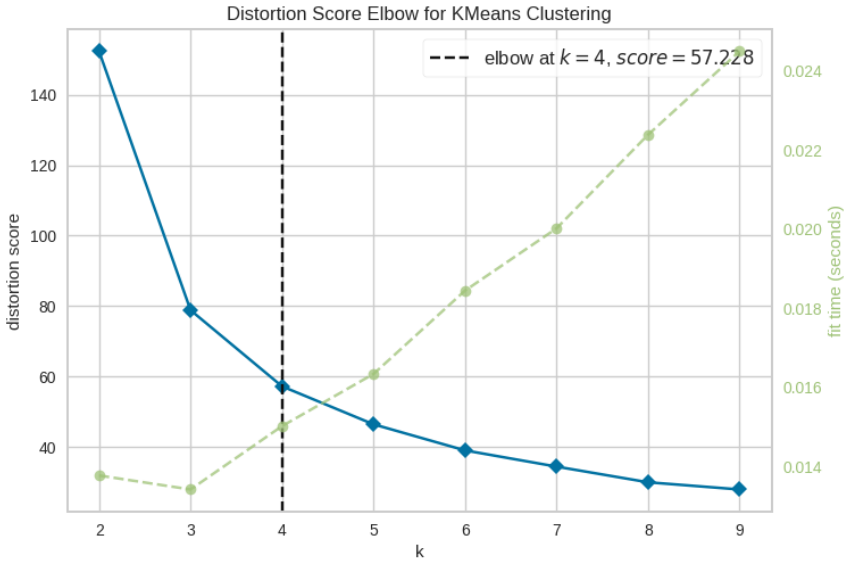
```
from sklearn.metrics import adjusted_rand_score
df['clusters'] = clusters

adjusted_rand_score(df['species'], df['clusters'])
```

```
0.7302382722834697
```

نسبتا خوبه

محاسبه حرفه ای تر Elbow

```
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans

k_means_yellowbrick = KMeans()
graph = KElbowVisualizer(k_means_yellowbrick, k=(2, 10))
graph.fit(df.iloc[:, 0:4])
graph.poof()
```



اینجا به ما عدد 4 رو نشون داده

یه بار K menas رو با 4 تا خوشه ران می کنیم. و جواب تستش

```
0.6498176853819967
```

مشخصه که سه تایی بهتر بود.

بزار اینم بهت لای پرانتز بگم، (که تعداد کلاسها توی لیبل های واقعی 3 تاست).

میتونیم به شکل زیر هم دقتش رو محاسبه کنیم.

```
from sklearn.metrics import silhouette_score

print("Silhouette score for n=3: ", silhouette_score(df.iloc[:, 0:4],
df['clusters']))z
```

```
Silhouette score for n=3 0.5528190123564102
```

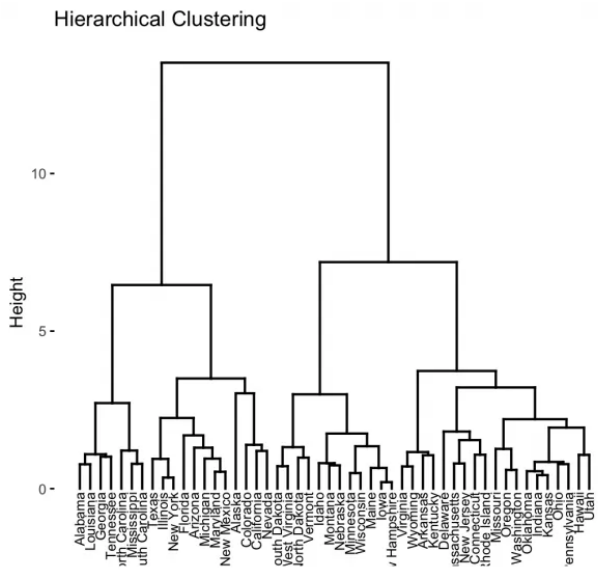
```
from sklearn.metrics import silhouette_score

print("Silhouette score for n=4: ", silhouette_score(df.iloc[:, 0:4],
df['clusters']))
```

```
Silhouette score for n=4:  0.49805050499728815
```

خوشه بندی سلسه مراتبی(Hierarchical Clustering)

خوشه بندی سلسه مراتبی یک روش یادگیری ماشینی بدون نظارت است که می‌توانید از آن برای پیش‌بینی زیرگروه‌ها بر اساس تفاوت بین نقاط داده و نزدیک‌ترین همسایگان آن‌ها استفاده کنید. همه نقاط داده بر اساس ماتریس فاصله ای که انتخاب می کنید به نزدیکترین همسایه خود مرتبط می شوند. نکته‌ای که این روش را نسبت به روش‌های دیگر خوشه‌بندی مجزا می‌کند، وجود ترتیب و یک نگاه از بالا به پایین (یا از پایین به بالا) است که در این تکنیک وجود دارد.

نمودار زیر نتیجه یک خوشه‌بندی سلسله مراتبی را به صورت «درختواره نگار» (Dendrogram) نشان می‌دهد.



خوشه‌بندی سلسله مراتبی با روش تجمیعی(Agglomertive): اگر دیدگاه به این نمودار از پایین به بالا باشد (Bottom-Up)، برحسب ارتفاع نمودار (Height) در سطح پایینی، خوشه‌ها زیرمجموعه خوشه‌های سطح بالاتر هستند در نتیجه به نظر می‌رسد که خوشه‌های زیرین با یکدیگر ترکیب شده و خوشه‌های سطح بالاتر را ایجاد می‌کنند. این شیوه خوشه بندی سلسله مراتبی به روش «تجمیعی» (Agglomerative) معروف است. برای نام‌گذاری این روش معمولا از عبارت اختصاری HAC که خلاصه Hierarchical Agglomertive Clustering است، استفاده می‌شود.

در الگوریتم HAC پیچیدگی زمانی برابر با $O(n^3)$ و فضای مورد نیاز حافظه نیز برابر با $O(n^2)$ است. بنابراین با افزایش حجم داده‌ها، سرعت و فضای حافظه برای اجرای عملیات خوشه‌بندی به شدت افزایش می‌یابد. به همین دلیل معمولا از این الگوریتم برای خوشه‌بندی «کلان داده» (Big Data) استفاده نمی‌شود.

خوشه‌بندی سلسله مراتبی با روش تقسیمی(Divisive): برعکس اگر دیدگاه از بالا به پایین (Top-Down) باشد، خوشه‌های بالایی به زیرخوشه‌ها دیگر تجزیه می‌شوند تا آنکه به خوشه‌هایی با تنها یک عضو برسیم. به این ترتیب بزرگترین خوشه که شامل همه مشاهدات است به کوچکترین خوشه‌ها که شامل تنها یک مشاهده است تقسیم می‌شود. به این روش «تقسیمی» (Divisive) گفته می‌شود. از آنجایی که این روش دارای محدودیت‌هایی است کمتر در خوشه‌بندی سلسله مراتبی به کار گرفته می‌شود.

روش های محاسبه فاصله یا Distance metrics

- اقلیدوسی (Euclidean)
- منهتن (Manhattan)
- کسینوسی (Cosine)

پارامترهای پیوند یا Linkage Parameters

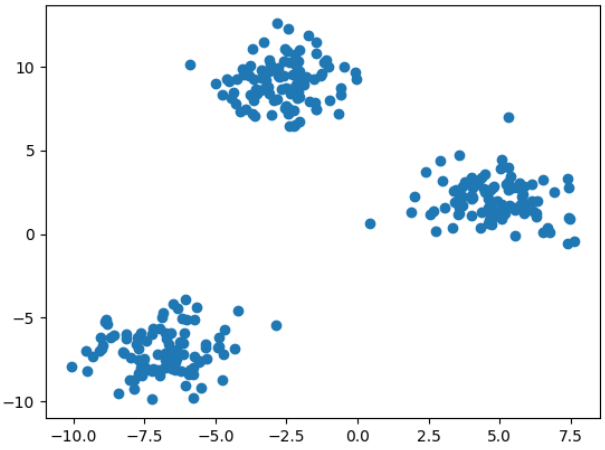
- Ward
- Compliter
- Avarage

بیاید یک دیتاست تصادفی بسازیم (ساخت دستی دیتاست، ساخت دیتاست)

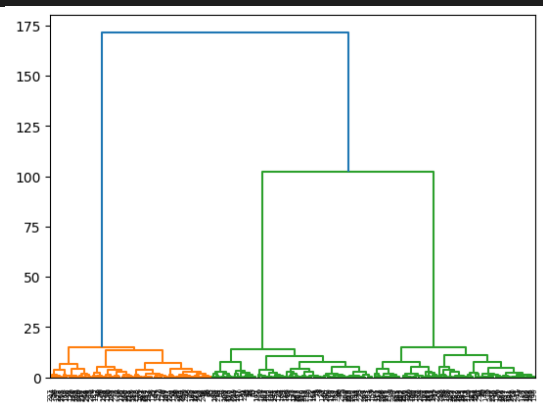
```
from sklearn.datasets import make_blobs
example_dataset = make_blobs(n_samples=300, n_features=2, centers=3,
cluster_std=1.3, random_state=42)
df = pd.DataFrame(columns=['A', "B"], data=example_dataset[0])
df['C'] = example_dataset[1]
df.head()
```

	A	B	C
0	-7.476796	-7.984907	2
1	-7.998165	-7.380032	2
2	-1.439889	7.427189	0
3	4.356893	3.401580	1
4	-9.529189	-8.190622	2

```
plt.scatter(df['A'], df['B'])
```




```
import scipy.cluster.hierarchy as sch
dendogram = sch.dendrogram(sch.linkage(df.iloc[:, 0:2], method='ward'))
```



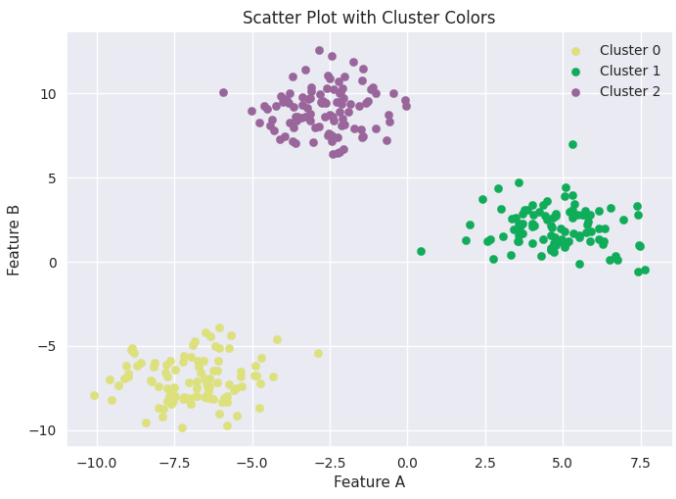
عملیات خوشه بندی

```
from sklearn.cluster import AgglomerativeClustering

hierarchical_clustering = AgglomerativeClustering(n_clusters = 3, affinity
= "euclidean", linkage = "ward")

y_hier_cluster = hierarchical_clustering.fit(df.iloc[:, 0:2])
clusters = y_hier_cluster.labels_
clusters
```

```
array([0, 0, 2, 1, 0, 1, 2, 1, 2, 2, 2, 1, 2, 2, 0, 2, 0, 1, 2, 2, 2, 2,
   1, 0, 2, 0, 0, 1, 1, 2, 2, 2, 0, 2, 0, 2, 0, 1, 0, 1, 1, 2, 0, 1,
   2, 2, 0, 1, 0, 1, 1, 0, 0, 2, 0, 1, 0, 2, 1, 2, 0, 1, 1, 0, 0, 1,
   1, 0, 0, 2, 1, 0, 0, 2, 2, 0, 0, 1, 2, 1, 2, 2, 0, 2, 1, 0, 0, 2,
   1, 2, 0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 1, 2, 1, 2, 2, 2, 2, 2, 1, 0,
   1, 2, 2, 2, 2, 1, 0, 1, 0, 1, 1, 1, 2, 0, 0, 0, 0, 2, 0, 0, 2, 2,
   2, 2, 2, 1, 1, 0, 2, 0, 2, 2, 0, 2, 1, 1, 1, 2, 1, 2, 2, 0, 1, 0,
   2, 1, 1, 0, 0, 2, 2, 0, 0, 0, 2, 0, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1,
   1, 2, 1, 1, 0, 2, 0, 1, 1, 0, 1, 2, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
   2, 0, 2, 2, 1, 1, 2, 1, 0, 0, 1, 2, 2, 0, 1, 1, 0, 0, 0, 0, 2, 0,
   0, 1, 0, 0, 2, 1, 0, 0, 1, 2, 2, 0, 2, 0, 1, 1, 0, 1, 0, 0, 0, 1,
   1, 2, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 2, 0, 2, 2, 2, 0, 2,
   1, 1, 0, 1, 1, 2, 2, 1, 1, 1, 0, 0, 0, 2, 2, 2, 1, 1, 1, 1, 0, 1,
   0, 1, 1, 0, 2, 1, 1, 2, 0, 2, 1, 2, 0, 0])
```

نمایش نمودار

```
import numpy as np

def random_color():
    return np.random.rand(3,)

for cluster_id in np.unique(clusters):
    cluster_points = df[clusters == cluster_id]
    plt.scatter(cluster_points['A'], cluster_points['B'],
c=random_color(), s=35, label=f"Cluster {cluster_id}")

plt.xlabel('Feature A')
plt.ylabel('Feature B')
plt.title('Scatter Plot with Cluster Colors')
plt.legend()

plt.show()
```



کاهش ابعاد (Dimensionality reduction)

کاهش ابعاد یا فروکاهی ابعاد (Dimension reduction) به فرایند کاستن و کم‌کردن از تعداد ابعاد و متغیرهای مورد نیاز برای نمایش و بررسی مسائل مطرح در ریاضیات، آمار، فیزیک، مهندسی، و بسیاری از شاخه‌های علوم محاسباتی و پیچیدهٔ نوین اطلاق می‌شود. در ادبیات تحلیل‌های چند متغیری اساساً به روش‌هایی که برای کاهش ابعاد استفاده می‌شود، روش‌های محوری یا روش‌های هندسی گفته می‌شود. کاهش ابعاد به دو دسته انتخاب ویژگی و استخراج ویژگی تقسیم می‌شود. در انتخاب ویژگی که در فضای اندازه‌گیری انجام می‌شود هدف پیدا کردن ویژگی‌های مطلوب از بین کل ویژگی‌های موجود است در حالی در استخراج ویژگی هدف انتقال ویژگی‌های انتخاب شده از فضای با ابعاد بیشتر به فضای با ابعاد کمتر و تعداد متغیرهای کمتر می‌باشد.

انتخاب ویژگی‌ها

در این نگرش به دنبال زیرمجموعه‌ای از متغیرهای اصلی مسئله (که ویژگی یا خصوصیت نیز نامیده می‌شوند.) هستیم که بتواند به درستی نمونه‌های مسئله را از هم تفکیک کند.

استخراج ویژگی

استخراج ویژگی (به انگلیسی: Feature extraction) فرایندی است که در آن داده‌ها در فضای با بعد بالا به فضای با بعد کمتر نگاشت می‌شوند. این نگاشت می‌تواند خطی (مانند روش تحلیل مؤلفه‌های اصلی(PCA)) یا غیر خطی باشد.

PCA (Principal Component Analysis)

تحلیل مولفه های اصلی: این روش اصلی‌ترین روش خطی برای کاهش ابعاد است؛ این روش نگاشت خطی داده‌ها را به یک فضا با بعد پایین‌تر انجام می‌دهد، به طوری که میزان توضیح واریانس داده اصلی در داده منتقل شده (به ابعاد کمتر) بیشینه باشد.

منطق اصلی پشت PCA نشان دادن داده های چند بعدی با متغیرهای کمتر با در نظر گرفتن ویژگی های اساسی است. PCA یک روش بسیار موثر برای آشکارسازی اطلاعات داده های ضروری است.

هدف اصلی PCA این است که یک فضای با ابعاد کمتر پیدا کند که بیشترین واریانس را در داده‌ها حفظ کند.(این جمله به معنی این است که این فرآیند سعی دارد تا حداکثر تنوع یا تغییرات موجود در داده‌ها را حفظ کند.) این به‌معنای یافتن مجموعه ای از محورهای جدید است که بیشترین میزان پراکندگی داده‌ها را در خود جای می دهند. این محورهای جدید به‌عنوان مؤلفه‌های اصلی یا PC ها شناخته می‌شوند. . PCA با کاهش ابعاد داده‌ها، پیچیدگی محاسباتی را کاهش می‌دهد و به بهبود کارایی الگوریتم‌های یادگیری ماشین کمک می‌کند. همچنین با تمرکز بر مؤلفه‌های اصلی، نویز موجود در داده‌ها را کاهش می‌دهد و به بهبود دقت الگوریتم‌های یادگیری ماشین کمک می‌کند. علاوه‌براین، PCA با کاهش ابعاد داده‌ها، تجسم داده‌ها در فضای با ابعاد کمتر را تسهیل می‌کند.

برای انجام‌دادن PCA این مرحله‌ها دنبال می‌شود:

1. داده‌ها را استانداردسازی کنید: داده‌ها را به‌گونه‌ای استاندارد کنید که میانگین هر ویژگی صفر و واریانس آن یک باشد.
2. کواریانس را محاسبه کنید: ماتریس کواریانس داده‌ها را محاسبه کنید.

3. ارزش‌های ویژه و بردارهای ویژه را محاسبه کنید: ارزش‌های ویژه و بردارهای ویژه ماتریس کواریانس را محاسبه کنید.
4. مهمترین PC ها را انتخاب کنید: مهمترین PC ها را با توجه به ارزش‌های ویژه آن‌ها انتخاب کنید.
5. داده‌ها را به فضای جدید تبدیل کنید: داده‌ها را به فضای با ابعاد کمتر تشکیل شده توسط PC های انتخابی تبدیل کنید.

با دنبال‌کردن این مرحله‌ها، PCA می‌تواند ابعاد داده‌ها را بدون ازدست‌دادن اطلاعات مهم کاهش دهد و به بهبود کارایی الگوریتم‌های یادگیری ماشین کمک کند.

پارامتر های PCA

- Components: محورهای اصلی در فضای ویژگی ها حداکثر جهت واریانس در داده ها را نشان می دهند. در حین تنظیم الگوریتم می توان تعداد Component ها را وارد کرد.
- Explained_variance: این مقدار واریانس توضیح داده شده توسط هر یک از مؤلفه های انتخاب شده را توصیف می کند.
- Explained_variance_ratio: این نشان دهنده درصد واریانس توضیح داده شده توسط هر یک از مؤلفه های انتخاب شده است.
- N_components: این به ما کمک می کند تا تعداد تخمینی component ها را مشخص کنیم.
- n_features: این فقط تعداد ویژگی ها یا متغیر های مستقل دیتاست ماست

مثال:

```
from sklearn.datasets import load_digits
digits = load_digits()

features = digits['data']
feature_names = digits["feature_names"]
labels = digits['target']

df = pd.DataFrame(data = features, columns = feature_names)
df.head()
```

فرض کنید دیتای بالا رو داریم که shape اون به صورت زیر است

```
(1797, 64)
```

حالا باید اون رو مقیاس دهی یا scale کنیم

میتونیم از دو روش زیر استفاده کنیم

Method 1:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaled_data = scaler.fit_transform(df)

scaled_df = pd.DataFrame(data=scaled_data, columns=feature_names)
scaled_df.head()
```

Method 2:

```
from sklearn.preprocessing import scale

scaled_data = scale(df)

scaled_df = pd.DataFrame(data=scaled_data, columns=feature_names)
scaled_df.head()
```

خروجی جفت روش ها یکی هست.

حالا وقت تبدیل 64 ستون به 4 ستون رسده

```
from sklearn.decomposition import PCA

pca = PCA(n_components=4)
pca_data = pca.fit_transform(scaled_df)

pca_data.shape
```

```
(1797, 4)
```

تبدیلش می کنیم به دیتافریم

```
component_df = pd.DataFrame(data = pca_data, columns=["first_component",
"second_component", "third_component", "fourth_component"])
```

```
pca.explained_variance_
```

```
array([7.34477606, 5.83549054, 5.15396115, 3.9662359 ])
```

```
pca.explained_variance_ratio_
```

```
array([0.12033916, 0.09561054, 0.08444415, 0.06498408])
```

این مقادیر از تجزیه و تحلیل اجزای اصلی (PCA) نشان‌دهنده‌ی واریانس توضیح داده‌ها در هر یک از اجزای اصلی هستند. دو مقدار زیر توضیح می‌دهند:

1. pca.explained_variance_:
   - این مقادیر واریانس توضیح داده‌ها را در هر یک از اجزای اصلی نشان می‌دهند.
   - در اینجا دو مقدار داریم: 7.34477606 و 5.83549054.
   - این مقادیر نشان‌دهنده‌ی میزان واریانسی است که توسط هر یک از اجزای اصلی توضیح داده‌ها توضیح داده می‌شود.
2. pca.explained_variance_ratio_:
   - این مقادیر نسبت واریانس توضیح داده‌ها در هر یک از اجزای اصلی را نشان می‌دهند.
   - در اینجا دو مقدار داریم: 0.12033916 و 0.09561054.
   - این مقادیر نشان‌دهنده‌ی میزان واریانسی است که توسط هر یک از اجزای اصلی توضیح داده‌ها توضیح داده می‌شود، به‌طور نسبی به کل واریانس توضیح داده‌ها.

با توجه به این مقادیر، می‌توانید ببینید که اجزای اصلی اول و دوم به ترتیب 12.03٪ و 9.56٪ از کل واریانس توضیح داده‌ها را توضیح می‌دهند.

برای explained_variance_ ، مقدار زیاد یک کامپوننت نشون دهنده این هست که اون کامپوننت نقش پررنگی در کاهش بعد ایفا میکنه و مقدار کم هم برعکس، به عبارت دیگر، ابعاد با مقادیر بالای explained_variance معمولاً توانایی توضیح دادن تغییرات داده‌ها را دارند. حالت ratio هم صرفا گزارش به صورت نسبت هست.

با توجه به این توضیحات، اگر مقادیر زیاد باشند، اجزا توانسته‌اند تغییرات زیادی در داده‌ها را توضیح دهند و این می‌تواند مفید باشد. اگر مقدار explained_variance_ratio برای یک از کامپوننت ها بالا باشد، این نشان‌دهنده‌ی اهمیت آن کامپوننت در حفظ اطلاعات تغییرات داده‌ها است.

```
np.cumsum(pca.explained_variance_ratio_)
```

```
array([0.12033916, 0.21594971, 0.30039385, 0.36537793])
```

:np.cumsum(pca.explained_variance_ratio_)

1. این مقدار نشان‌دهنده‌ی تجمعی نسبت واریانس توضیح داده‌ها توسط اجزا است.
2. اگر مقدار آنها زیاد باشد، خوب است. این نشان‌دهنده‌ی میزان کل واریانس توضیح داده‌ها که توسط اجزا توضیح داده می‌شود، است.

Recommender systems

ریکامندر سیستم یا سیستم پیشنهاد دهنده هموطنور که از اسمش مشخصه، با توجه به اطلاعاتی که توی یه مجموعه بزرگی از داده ها داره میاد یه پیشنهادی به کاربر میده، برای مثال میشه به سیستم های پیشنهاده دهنده Netflix و Imdb و ... اشاره کرد. این سیستم های پیشنهاد دهنده دو نوع مختلف دارند. (من خودم زیاد از این بخش خوشم نمیاد).

1. Content-Based (مبتنی بر محتوا):
   a. این روش میگه بیا چیز هایی رو به من نشون بده که شبیه به چیز هایی هستند که من دوست داشته ام.

2. Collaborative Filtering (فیلتر مشارکتی): بیا چیز هایی رو به من نشون بده که افراد مشابه به من دوست دارند.

Content Based

فرض کنید فیلم های زیر را با تگ هاشون داریم. و یک کاربر به سه تای اول آنها امتیاز داده باشد

Batman vs Superman: Adventure, Super hero (rate: 2)

Guardians of Galaxy: Comedy, Adventure, Super hero, Sci-Fi (rate: 10)

Capitan America: Comedy, Super hero (rate: 8)

Hitchhiker's guide to the galaxy: Comedy, Adventure, Super hero

Batman begins: Super hero

Spiderman: Comedy, Super hero

Input user Ratings

Batman vs Superman	2
Guardians of Galaxy	10
Capitan America	8

Movies Matrix

Comedy	Adventure	Super hero	Sci-Fi
0	1	1	0
1	1	1	1
1	0	1	0

حالا دوتا ماتریس بالا را توی هم ضرب می کنیم تا ماتریس وزن دهی شده ژانر ها را بدست آوریم.

Weighted Genre Matrix:

Comedy	Adventure	Super hero	Sci-Fi
0	2	2	0
10	10	10	10
8	0	8	0

حالا با نرمالسازی به صورت تقسیم امتیاز هر فیلم به جمع کل امتاز ها که 60 هست ماتریس زیر بدست می آید.

User Profile Matrix:

Comedy	Adventure	Super hero	Sci-Fi
$18 \div 60 = 0.3$	$12 \div 60 = 0.2$	$20 \div 60 = 0.33$	$10 \div 60 = 0.16$

حالا میریم سراغ سه فیلم بعدی که کاربر اونا رو ندیده

Movies Matrix:

	Comedy	Adventure	Super hero	Sci-Fi
Hitchhiker's guide to the galaxy	1	1	0	1
Batman begins	0	0	1	0
Spiderman	1	0	1	0

Weighted Movies Matrix

	Comedy	Adventure	Super hero	Sci-Fi
Hitchhiker's guide to the galaxy	0.3	0.2	0	0.16
Batman begins	0	0	0.33	0
Spiderman	0.3	0	0.33	0

حالا امتیاز هر فیلم را جمع می مکنیم

Hitchhiker's guide to the galaxy: 0.3 + 0.2 + 0.16 = 0.66

Batman begins: 0.33

Spiderman: 0.3 + 0.33 = 0.63

حالا مشخص شد که فیلم Batman begins از همه کمتر مورد پسند این کاربر خواهد بود و فیلم Hitchhiker's guide to the galaxy از بقیه نزدیک تر به سلیقه کاربر ما خواهد بود.

بریم یه پروژه رو با هم انجام بدیم.

میتونید دیتاست رو به صورت زیر دانلود کنید: (ولی من هم داخل گیتهاب گذاشتمشون، از طریق لینک صفحه اول می‌تونید دانلودش کنید).

```
# Get the Dataset
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-
SkillsNetwork/labs/Module%205/data/moviedataset.zip
print('unziping ...')
!unzip -o -j moviedataset.zip
```

کتابخانه های مورد نیاز

```
#Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only
that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

خواندن دیتاست

```
movies_df = pd.read_csv('movies.csv')
movies_df.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

توی این پروژه کوچیک میخوایم ببینیم هرکس چه فیلم ها و ژانر هایی رو دیده و چه امتیاز هایی رو بهشون داده و با توجه به اون بهش فیلم های دیگه ای رو پیشنهاد بدیم.

اولین کاری که باید بکنیم اینه که سال انتشار فیلم هارو از توی قسمت اسم هاشون حذف کنیم (پس از رجکس استفاده می کنیم).

```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have
years in their titles
movies_df['year'] =
movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace(r'\s\(\d{4}\)', '',
regex=True)
#Applying the strip function to get rid of any ending whitespace
characters that may have appeared
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

	movieId	title	genres	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995
1	2	Jumanji	Adventure|Children|Fantasy	1995
2	3	Grumpier Old Men	Comedy|Romance	1995
3	4	Waiting to Exhale	Comedy|Drama|Romance	1995
4	5	Father of the Bride Part II	Comedy	1995

حالا میایم ژانر هارو که با | جدا شدن، به صورت لیست در میاریم.

```
#Every genre is separated by a | so we simply have to call the split
function on |
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

مرحله بعدی

```
#Copying the movie dataframe into a new one since we won't need to use the
genre information in our first case.
moviesWithGenres_df = movies_df.copy()

#For every row in the dataframe, iterate through the list of genres and
place a 1 into the corresponding column
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        moviesWithGenres_df.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that
column's genre
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

توی کد بالا گفته که

حلقه اول: به ازای تک تک سطر ها

حلقه دوم: به ازای تک تک ژانر ها توی سطر انتخاب شده

moviesWithGenres_df.at[index, genre] = 1 ، این کد داره میگه که توی دیتافریم جدید مقدار ژانر رو سطر index برابر با 1 بزار.

حالا این کار ممکنه باعث این بشه که توی یکی از سطر ها یه ژانر اومده باشه که توی سطر های قبلی نبوده، پس وقتی این سطر جدید ساخته بشه تمام مقادیر اون برای سطر های قبلی خالی میشه، که ما این مقادیر خالی رو با عدد صفر پر کردیم توی خط یکی مونده به آخر.

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...	Horror	Mystery	Sci-Fi	IMAX	Documentary	War	Musical	Western	Film-Noir	(no genres listed)
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3	Grumpier Old Men	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	0.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5	Father of the Bride Part II	[Comedy]	1995	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 24 columns

حالا فرض کنید یه یوزری داریم که این فیلم هارو دیده و بهشون نظر داده

```
userInput = [
             {'title':'Breakfast Club, The', 'rating':5},
             {'title':'Toy Story', 'rating':3.5},
             {'title':'Jumanji', 'rating':2},
             {'title':"Pulp Fiction", 'rating':5},
             {'title':'Akira', 'rating':4.5}
           ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5

حالا میخوایم به دیتاستمون mobieID رو هم اضافه کنیم

```
#Filtering out the movies by title
inputId =
movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by
title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop(['genres'], axis = 1).drop(['year'], axis =
1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the
original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

	movieId	title	rating
0	1	Toy Story	3.5
1	2	Jumanji	2.0
2	296	Pulp Fiction	5.0
3	1274	Akira	4.5
4	1968	Breakfast Club, The	5.0

حالا کاری که میکنیم اینه که از توی دیتاست تمام فیلم هامون میایم فقط اونایی رو انتخاب می‌کنیم که کاربر ما دیده

```
#Filtering out the movies from the input
userMovies =
moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movie
Id'].tolist())]
userMovies
```

	movieId	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...	Horror	Mystery	Sci-Fi	IMAX	Documentary	War	Musical	Western	Film-Noir	(no genres listed)
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	0	1.0	1.0	1.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2	Jumanji	[Adventure, Children, Fantasy]	0	1.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
293	296	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1246	1274	Akira	[Action, Adventure, Animation, Sci-Fi]	0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1885	1968	Breakfast Club, The	[Comedy, Drama]	0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

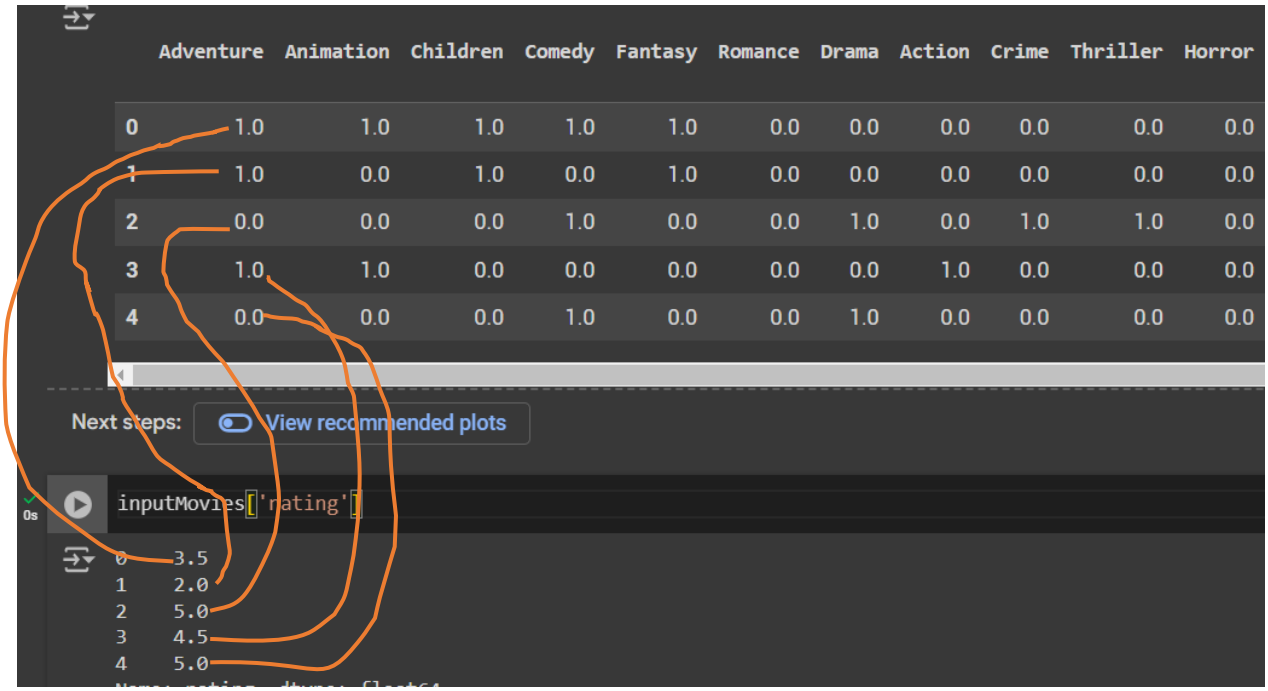
5 rows × 24 columns

حالا یکم تروتمیز میکنیم دیتاستمون رو

```
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop(['movieId', 'title', 'genres', 'year'],
axis = 1)
userGenreTable
```

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	IMAX	Documentary	War	Musical
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

حالا کاری که باید بکنیم اینه که امتیاز هایی که کاربر داده به فیلم هارو توی ماتریس ژانر ها ضرب کنیم، تا بفهمیم به هر ژانر چقدر علاقه داره. و یوزر پروفایلش ساخته بشه.



	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Next steps: View recommended plots

```
inputMovies['rating']
```

```
0    3.5
1    2.0
2    5.0
3    4.5
4    5.0
```

```
#Dot produt to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

```
Adventure           10.0
Animation            8.0
Children             5.5
Comedy              13.5
Fantasy              5.5
Romance              0.0
Drama               10.0
Action               4.5
Crime                5.0
Thriller             5.0
Horror               0.0
Mystery              0.0
Sci-Fi               4.5
IMAX                 0.0
Documentary          0.0
War                  0.0
Musical              0.0
Western              0.0
Film-Noir            0.0
(no genres listed)   0.0
dtype: float64
```

حالا بیاید همین کار هارو روی کل دیتاست فیلم ها انجام بدیم تا ببینیم کدوم فیلم رو چقدر دوس داره

```
#Now let's get the genres of every movie in our original dataframe
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop(['movieId', 'title', 'genres', 'year'], axis
= 1)
genreTable.head()
```

movieId	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	IMAX	Documentary	War
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

حالا ماتریس ژانر هارو توی یوزر پروفایل ضرب می کنیم و ارزش ژانر هارو برای هر فیلم جمع میکنیم و بر جمع مقادیر یوزر پروفایل تقسیم میکنیم تا ارزش هر فیلم در بیاد

```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df =
((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head()
```

```
movieId
1    0.594406
2    0.293706
3    0.188811
4    0.328671
5    0.188811
dtype: float64
```

حالا مقادیر بالا رو مرتب می کنیم تا ببینیم کدوم فیلم بیشترین امتیاز رو میاره

```
#Sort our recommendations in descending order
recommendationTable_df =
recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

```
movieId
5018      0.748252
26093     0.734266
27344     0.720280
148775    0.685315
6902      0.678322
dtype: float64
```

مشخصه که فیلم 5018 بیشترین امتیاز رو آورده

حالا میخوایم اسم 20 تا فیلم برتری که این کاربر ممکنه خوشش بیاد رو نمایش میدیم

```
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

	movieId	title	genres	year
664	673	Space Jam	[Adventure, Animation, Children, Comedy, Fanta...	NaN
1824	1907	Mulan	[Adventure, Animation, Children, Comedy, Drama...	NaN
2902	2987	Who Framed Roger Rabbit?	[Adventure, Animation, Children, Comedy, Crime...	NaN
4923	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	NaN
6793	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	NaN
8605	26093	Wonderful World of the Brothers Grimm, The	[Adventure, Animation, Children, Comedy, Drama...	NaN
8783	26340	Twelve Tasks of Asterix, The (Les douze travau...	[Action, Adventure, Animation, Children, Comed...	NaN
9296	27344	Revolutionary Girl Utena: Adolescence of Utena...	[Action, Adventure, Animation, Comedy, Drama, ...	NaN
9825	32031	Robots	[Adventure, Animation, Children, Comedy, Fanta...	NaN
11716	51632	Atlantis: Milo's Return	[Action, Adventure, Animation, Children, Comed...	NaN
11751	51939	TMNT (Teenage Mutant Ninja Turtles)	[Action, Adventure, Animation, Children, Comed...	NaN
13250	64645	The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thri...	NaN
16055	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	NaN
18312	91335	Gruffalo, The	[Adventure, Animation, Children, Comedy, Drama]	NaN
22778	108540	Ernest & Célestine (Ernest et Célestine)	[Adventure, Animation, Children, Comedy, Drama...	NaN

توجه کنید که ترتیب لیست بالا براثاث شماره فیلم هست نه بر اثاث علاقه مندی کاربر.

Collaborative Filtering

توی این بخش بیشتر روش های آماری درست می کنیم تا ببینیم یوزر ها چقدر به هم شبیه هستند. تا چیزایی که دوست دارن رو به هم دیگه نشون بدیم.

حالا این روش خودش به دو بخش تقسیم میشه دوباره

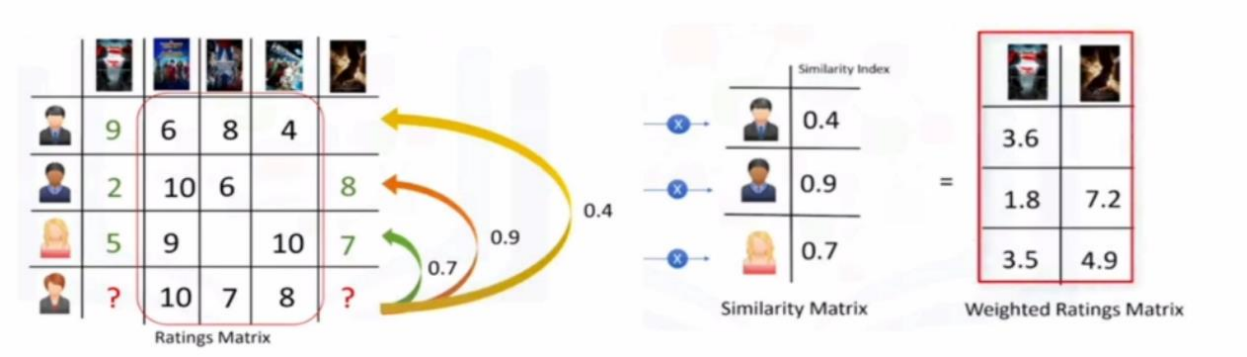
- User Based
  - شباهت یه یوزر نسبت به بقیه با توجه به رفتاری که از خودش نشون داده پیدا می کنیم.
- Item-Based
  - شباهت آیتم هارو پیدا میکنیم.

مشکلات Collaborative filtering:

- Data Sparsity:
  - ممکنه یوزر ها زیاد باشه، ولی امتیاز های زیادی نداده باشند.
- Cold Start:
  - وقتی یه کاربر تازه اومده و به هیچ چیزی هیچ امتیازی نداده باشه و یا ندیده باشه یا نخریده باشه، پس ما چطور می تونیم بهش چیز جدیدی پیشنهاد بدیم؟
- Scalability:
  - وقتی تعداد آیتم ها و یا یوزر ها زیاد میشه، ماتریس های ما خیلی بزرگ میشه و محاسبات بسیار سخت میشه

ولی همه مشکلات بالا راه حل هایی رو دارن، برای مثال یکی از راه حل ها اینه که از هر دو روش content based و Collaborative باهم استفاده کنیم.

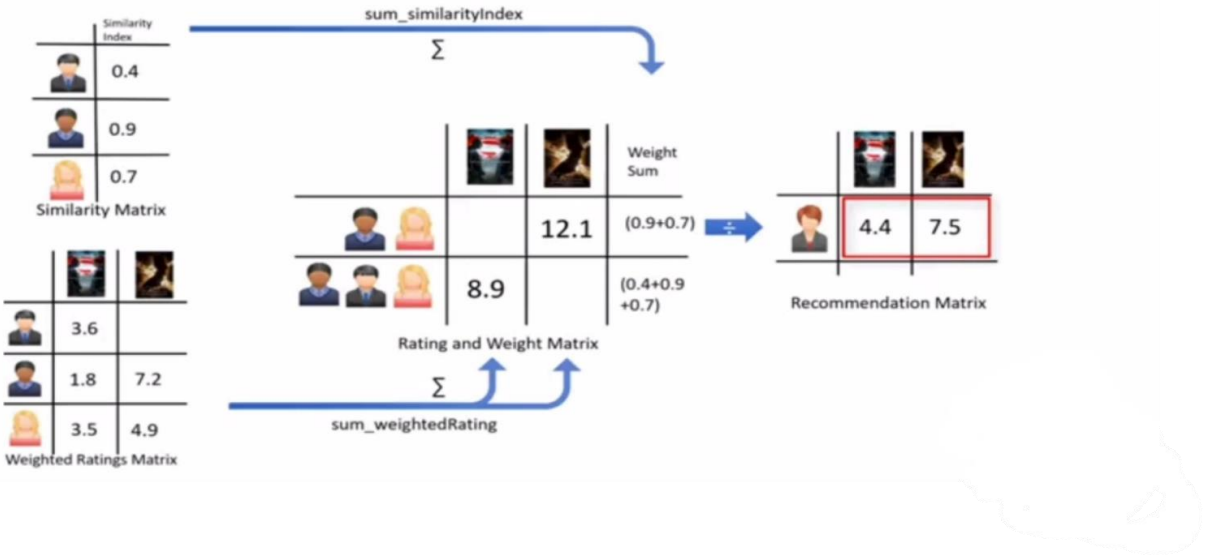
بریم سراغ User Based



فرض کنید کاربر های بالا رو داریم، میخوایم به کاربر چهارم براساس میزان شباهتش به کاربر های دیگه فیلم معرفی کنیم.

اون سه تا کاربر اول فیلم هارو دیدن و بهشون امتیاز دادن، و یوزر چهارم هم فقط 3 از اون 5 تا فیلمو دیده.

فرض کنید ماتریس شباهت را ما از یه طریقی پیدا کردیم و داریم، حالا میزان شباهت هر کاربر رو توی امتیازی که به فیلم ها داده ضرب می کنیم.



حالا امتیاز های فیلم هارو جمع می کنیم و در نهایت بر روی جمع شباهت کاربرانی که به انها امتیاز دادن تقسیم می کنیم و در نهایت ارزشی که کاربر چهارم برای این دوتا فیلم در نظر می گیره رو بدست می آوریم.

بریم یه پروژه بزنیم.

همون کارهای قبلی رو انجام میدیم برای پاکسازی داده ها، فقط این دفعه یه دیتاست دیگه هم به اسم ratings داریم که جلوتر میبینیمش.

```
#Dropping the genres column
movies_df = movies_df.drop(['genres'], axis = 1)
```

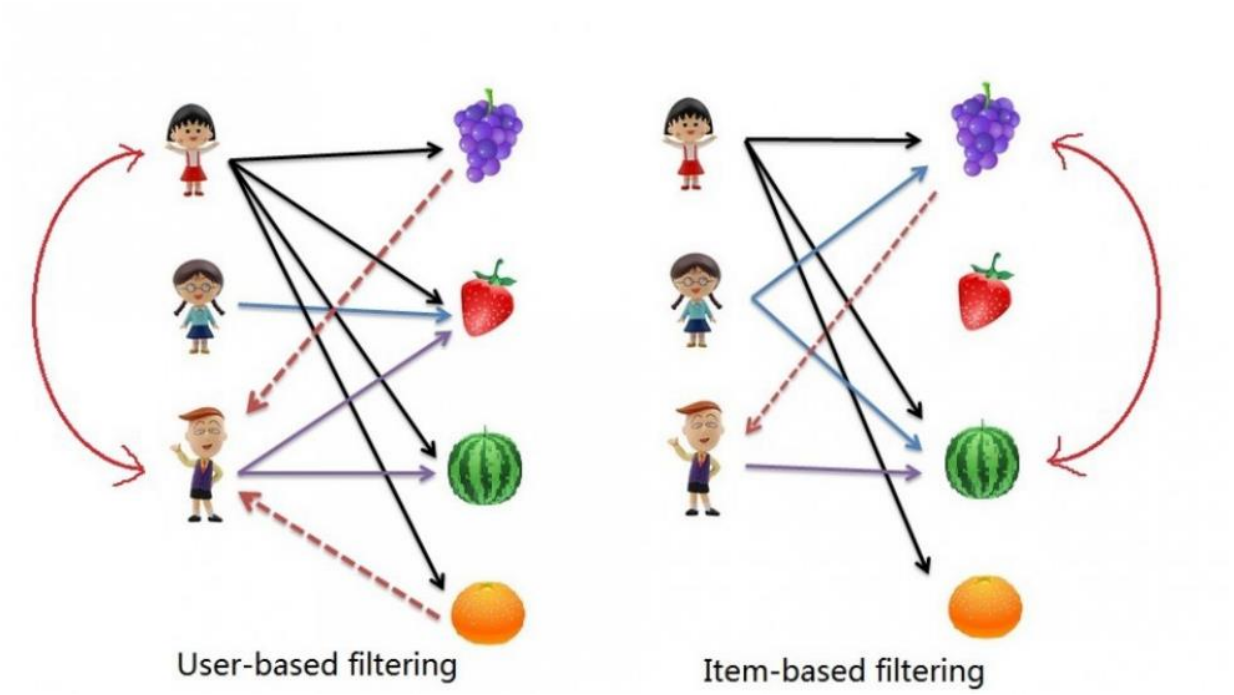
توی این روش کاری به ژانر فیلم ها نداریم پس پاکش می کنیم.

```
ratings_df = pd.read_csv('ratings.csv')
ratings_df.head()
```

	userId	movieId	rating	timestamp
0	1	169	2.5	1204927694
1	1	2471	3.0	1204927438
2	1	48516	5.0	1204927435
3	2	2571	3.5	1436165433
4	2	109487	4.0	1436165496

```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', axis = 1)
```

تایم رو پاک کردیم



یک یوزر تصادفی مسیازیم

```
userInput = [
            {'title':'Breakfast Club, The', 'rating':5},
            {'title':'Toy Story', 'rating':3.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5

حالا id فیلم هارو هم بهش اضافه کنیم

```
#Filtering out the movies by title
inputId =
movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by
title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop(['year'], axis = 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the
original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

حالا دنبال یوزر هایی می‌گردیم که این فیلم هارو دیده باشن

```
#Filtering out users that have watched movies that the input has watched
and storing it
userSubset =
ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
userSubset.head()
```

	userId	movieId	rating
19	4	296	4.0
441	12	1968	3.0
479	13	2	2.0
531	13	1274	5.0
681	14	296	2.0

حالا دیتافریم بالا رو بر اساس id گروهبندی می کنیم

```
#Groupby creates several sub dataframes where they all have the same value
in the column specified as the parameter
userSubsetGroup = userSubset.groupby(['userId'])
```

```

userSubsetGroup.get_group(1130)
```

	userId	movieId	rating
104167	1130	1	0.5
104168	1130	2	4.0
104214	1130	296	4.0
104363	1130	1274	4.5
104443	1130	1968	4.5

حالا بر اساس تعداد فیلم های مشترک یوزر هارو مرتب می کنیم.

```
#Sorting it so users with movie most in common with the input will have priority
userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True)
```

سه تای اولشو ببینیم

```
userSubsetGroup[0:3]
```

```
[((75,),
       userId  movieId  rating
  7507      75        1     5.0
  7508      75        2     3.5
  7540      75      296     5.0
  7633      75     1274     4.5
  7673      75     1968     5.0),
 ((106,),
       userId  movieId  rating
  9083     106        1     2.5
  9084     106        2     3.0
  9115     106      296     3.5
  9198     106     1274     3.0
  9238     106     1968     3.5),
 ((686,),
        userId  movieId  rating
  61336     686        1     4.0
  61337     686        2     3.0
  61377     686      296     4.0
  61478     686     1274     4.0
  61569     686     1968     5.0)]
```

حالا صدتا کاربر اولی که فیلم های مشترک با کاربر ما دارن رو انتخاب می کنیم

```
userSubsetGroup = userSubsetGroup[0:100]
```

کد زیر را اجرا می کنیم تا میزان شباهت هر یوزر را به کاربر خودمون محاسبه کنیم

```
#Store the Pearson Correlation in a dictionary, where the key is the user Id and the value is the coefficient
pearsonCorrelationDict = {}
```

```
#For every user group in our subset
for name, group in userSubsetGroup:
    #Let's start by sorting the input and current user group so the values
aren't mixed up later on
    group = group.sort_values(by='movieId')
    inputMovies = inputMovies.sort_values(by='movieId')
    #Get the N for the formula
    nRatings = len(group)
    #Get the review scores for the movies that they both have in common
    temp_df =
inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]
    #And then store them in a temporary buffer variable in a list format
to facilitate future calculations
    tempRatingList = temp_df['rating'].tolist()
    #Let's also put the current user group reviews in a list format
    tempGroupList = group['rating'].tolist()
    #Now let's calculate the pearson correlation between two users, so
called, x and y
    Sxx = sum([i**2 for i in tempRatingList]) -
pow(sum(tempRatingList),2)/float(nRatings)
    Syy = sum([i**2 for i in tempGroupList]) -
pow(sum(tempGroupList),2)/float(nRatings)
    Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) -
sum(tempRatingList)*sum(tempGroupList)/float(nRatings)

    #If the denominator is different than zero, then divide, else, 0
correlation.
    if Sxx != 0 and Syy != 0:
        pearsonCorrelationDict[name] = Sxy/sqrt(Sxx*Syy)
    else:
        pearsonCorrelationDict[name] = 0
```

```
pearsonCorrelationDict
```

```
{(75,): 0.8272781516947562,
 (106,): 0.5860090386731182,
 (686,): 0.8320502943378437,
 (815,): 0.5765566601970551,
 (1040,): 0.9434563530497265,
 (1130,): 0.2891574659831201,
 (1502,): 0.8770580193070299,
 (1599,): 0.4385290096535153,
 (1625,): 0.716114874039432,
 (1950,): 0.179028718509858,
 (2065,): 0.4385290096535153,
 (2128,): 0.5860090386731196,
 (2432,): 0.1386750490563073,
 (2791,): 0.8770580193070299,
 (2839,): 0.8204126541423674,
 (2948,): -0.11720180773462392,
 (3025,): 0.45124262819713973,
 (3040,): 0.89514359254929,
 (3186,): 0.6784622064861935,
 (3271,): 0.26989594817970664,
 (3429,): 0.0,
```

تبدیلش میکنیم به دیتافریم

```
# Create separate arrays for similarityIndex and userId
similarityIndex = list(pearsonCorrelationDict.values())
userId = [item[0] for item in pearsonCorrelationDict.keys()]

# Create the DataFrame
pearsonDF = pd.DataFrame({'similarityIndex': similarityIndex, 'userId':
userId})

# Display the first few rows
pearsonDF.head()
```

	similarityIndex	userId
0	0.827278	75
1	0.586009	106
2	0.832050	686
3	0.576557	815
4	0.943456	1040

مرتب کردن دیتاست

```
topUsers=pearsonDF.sort_values(by='similarityIndex',
ascending=False)[0:50]
topUsers.head()
```

	similarityIndex	userId
64	0.961678	12325
34	0.961538	6207
55	0.961538	10707
67	0.960769	13053
4	0.943456	1040

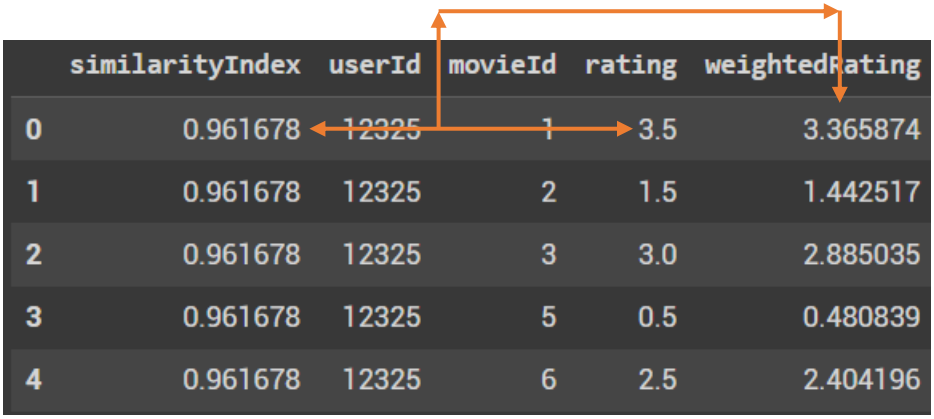
حالا مشخص می کنیم این کاربرا به فیلم ها چه امتیاز هایی دادن

```
topUsersRating=topUsers.merge(ratings_df, left_on='userId',
right_on='userId', how='inner')
topUsersRating.head()
```

	similarityIndex	userId	movieId	rating
0	0.961678	12325	1	3.5
1	0.961678	12325	2	1.5
2	0.961678	12325	3	3.0
3	0.961678	12325	5	0.5
4	0.961678	12325	6	2.5

حال میایم میزان امتیاز هر کاربر به فیلمش رو در میزان شباهت هر کاربر ضرب میکنیم و مشخص میکنیم که کاربر ما به چه احتمالی این فیلم هارو دوست خواهد داشت.

```
#Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] =
topUsersRating['similarityIndex']*topUsersRating['rating']
topUsersRating.head()
```



	similarityIndex	userId	movieId	rating	weightedRating
0	0.961678	12325	1	3.5	3.365874
1	0.961678	12325	2	1.5	1.442517
2	0.961678	12325	3	3.0	2.885035
3	0.961678	12325	5	0.5	0.480839
4	0.961678	12325	6	2.5	2.404196

حالا میزان وزن امتیاز های هر فیلم رو و میزان شباهتشون رو به صورت جداگانه جمع میکنیم

```
#Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating =
topUsersRating.groupby('movieId').sum()[['similarityIndex','weightedRating
']]
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

movieId	sum_similarityIndex	sum_weightedRating
1	38.376281	140.800834
2	38.376281	96.656745
3	10.253981	27.254477
4	0.929294	2.787882
5	11.723262	27.151751

حالا میزان شباهت هر فیلم رو بر وزن امتیازات همون فیلم تقسیم می کنیم تا بهترین فیلم هارو پیدا کنیم.

```
#Creates an empty dataframe
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] =
tempTopUsersRating['sum_weightedRating']/tempTopUsersRating['sum_similarit
yIndex']
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df.head()
```

movieId	weighted average recommendation score	movieId
1	3.668955	1
2	2.518658	2
3	2.657941	3
4	3.000000	4
5	2.316058	5

حالا همین لیست بالا رو مرتب می کنیم تا بهترین فیلم ها بیان بالا

```
recommendation_df = recommendation_df.sort_values(by='weighted average
recommendation score', ascending=False)
recommendation_df.head(10)
```

movieId	weighted average recommendation score	movieId
5073	5.0	5073
3329	5.0	3329
2284	5.0	2284
26801	5.0	26801
6776	5.0	6776
6672	5.0	6672
3759	5.0	3759
3769	5.0	3769
3775	5.0	3775
90531	5.0	90531

کاری که تا الان انجام دادیم کاملا دستی بوده پس تعجبی نداره که یکم پیچیده باشه، کتابخانه هایی هم هستند که برخی از این کارها رو برای ما راحت تر کنن.

منابع

www.geeksforgeeks.org

www.placabi.com

www.blog.faradars.org

www.scikit-learn.org

www.chistio.ir

www.cafetadris.com

www.wikipedia.org

www.howsam.org

www.maktabkhooneh.org: Jadi Machine Learning Course

www.udemy.com: Data Science Course

www.copilot.microsoft.com