

دانشگاه صنعتی خواجه نصیرالدین طوسی

گزارش مبانی سیستم های هوشمند

۴۰۱۲۱۹۹۳ ارشیا کلانتریان

۴۰۲۲۰۹۰۳ محمد حسین گل محمدی

مینی پروژه اول

پاییز ۱۴۰۴

سوال ۱ و ۲

25 / May / 2021

۱۱۴۴۰ / شوال / ۱۴

$$\text{specificity} = \frac{TN}{TN+FP}$$

$$\text{sensitivity} = \frac{TP}{TP+FN}$$

$$\text{sensitivity} = \frac{q_0}{q_0+q_1} = \frac{q_1}{q_1+V} = \frac{1-V}{1-V+V} = \frac{1-V}{1+V}$$

$$\text{sensitivity} = \frac{t_0}{t_0+V} = \frac{t_1}{t_1+I} = \frac{V}{V+I} = \frac{t_0}{t_0+I}$$

$$C^+ = \{(1, 1), (0, 1), (1, 0), (1, 1, 0), (1, 1, 1), (1, 1, 0, 1), (1, 1, 1, 1)\}$$

$$C^- = \{(-1, -1), (0, -1), (-1, 0), (-1, -1, 0), (-1, -1, 1), (-1, -1, 0, 1), (-1, -1, 1, 1)\}$$

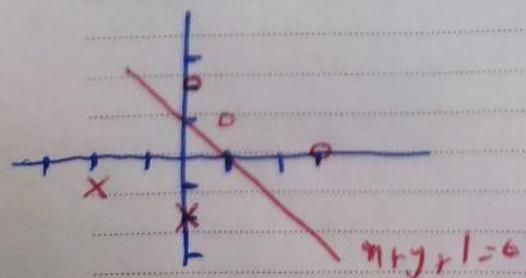
اون رست بجه آنیت کنیم

$$C_i^+ \rightarrow (0, 0, 0) \cdot (1, 1, 1) = 0 \xrightarrow{H=1} (1, 1, 1) = 1$$

$$C_i^+ = (1, 1, 1) \cdot (1, 1, 1, 1) > 0 \quad \checkmark \quad C_i^- = (1, 1, 1) \cdot (0, 1, 1, 1) > 0 \quad \checkmark$$

$$C_i^+ = (1, 1, 1) \cdot (1, 1, 1, 1) > 0 \quad \checkmark \quad C_i^- = (-1, -1, -1) \cdot (1, 1, 1, 1) > 0 \quad \checkmark$$

$$C_i^+ = (1, 1, 1) \cdot (1, 1, 1, 1) > 0 \quad \checkmark \quad C_i^- = (-1, -1, -1) \cdot (1, 1, 1, 1) > 0 \quad \cancel{\checkmark}$$



روز دزفول - روز مقاومت و پایداری

ش	س	ج	۲
۷	۶	۳	۱
۵	۴		
۱۴	۱۳	۱۲	۱۰

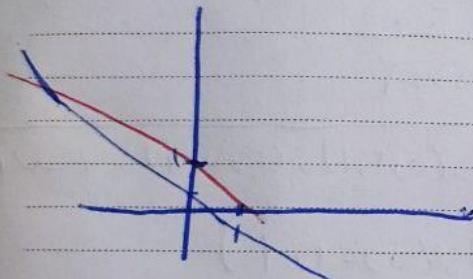
چهارشنبه
۱۴۰۰ خرداد
26 / May / 2021
۱۴۰۰ / شنبه / ۱۶

۵

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ -1 & -1 & 1 \\ 0 & -1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

$$(X^T X)^{-1} X^T y = (0.33, 0.33, 0.33)^T$$

$$0.33V_1 + 0.33V_2 + 0.33V_3$$



$$\underline{\mu}_+ = (-1, 1, 1) \quad \underline{\mu}_- = (1, -1, 1)$$

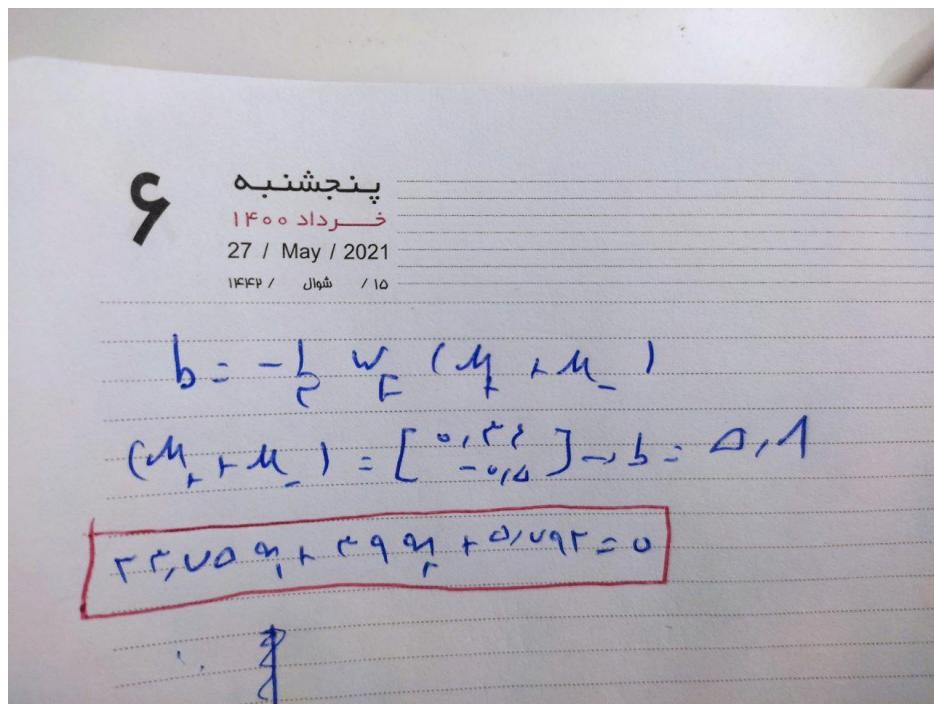
$$S_+ = \sum_{n \in S_+} (n - \underline{\mu}_+) (n - \underline{\mu}_+)^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$S_- = \sum_{n \in S_-} (n - \underline{\mu}_-) (n - \underline{\mu}_-)^T = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

$$S_W = S_+ + S_- = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

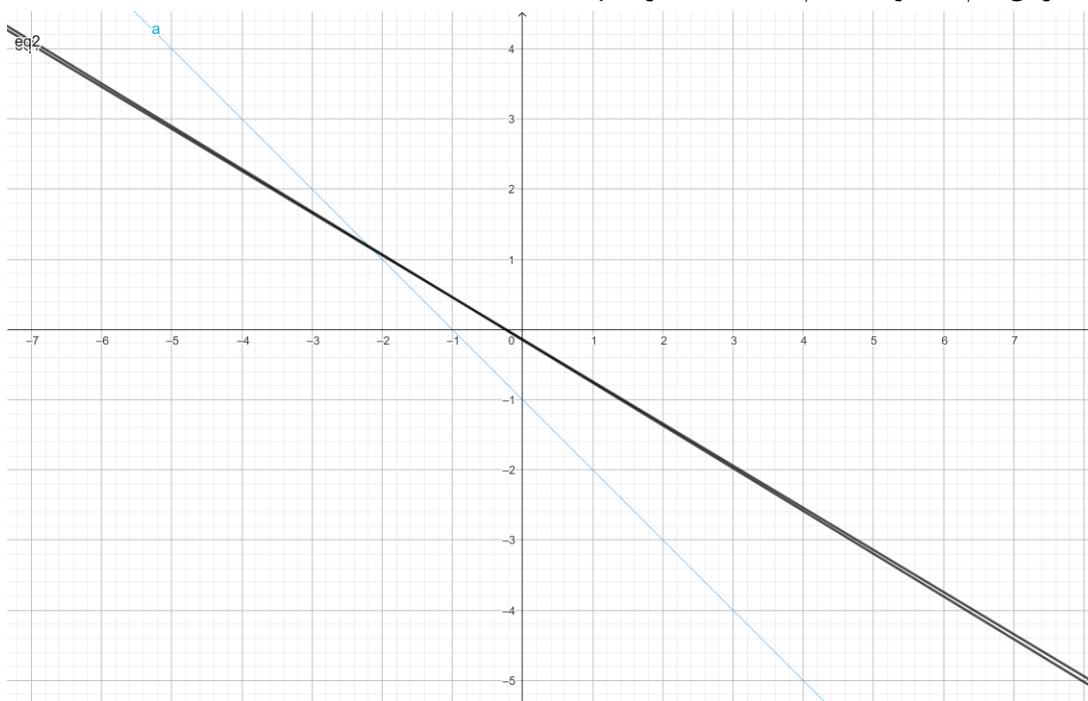
$$W_F = S_W^{-1} (\underline{\mu}_+ + \underline{\mu}_-) \quad S_W^{-1} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \quad \underline{\mu}_+ + \underline{\mu}_- = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$W_F = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad g(\text{cm}) = W_F n + b$$



-۳-۲

روش فیشر و کمترین مربعات تقریباً یک خط شدند و جداسازی بهتری به ما میدهند در mlp به دلیل اینکه فقط یک بار ان هم با میو ۱ انجام شده جدا سازی بهینه نیست



سوال ۳

- ۱-۳

چون pca بپر مبنای کوواریانس کار میکند چون واریانس λ_1 تقریباً ۱۰ برابر واریانس λ_2 هست تقریباً پراکنده‌گی به طور کامل در راستای λ_1 قرار خواهد داشت (تقریباً خط صاف میشه)

- ۲-۳

گرویزگی‌ها در مقیاس‌های بسیار متفاوت باشند، کوواریانس ویژگی با دامنه‌ی بزرگ‌تر مقدار عددی خیلی بزرگ‌تری خواهد داشت \rightarrow مقدار ویژه‌ی بزرگ‌تر PCA \rightarrow آن را مهم‌تر می‌داند.

بنابراین PCA جهت مؤلفه اصلی را تقریباً در راستای ویژگی بزرگ‌تر (در اینجا λ_1 انتخاب می‌کند)، حتی اگر از نظر «اطلاعاتی» هر دو ویژگی مهم باشند.

- ۳-۳

مرکز کردن داده‌ها: قبل از اجرای PCA باید مقدار میانگین هر ویژگی را از خودش کم کنیم تا داده‌ها دور محور صفر قرار بگیرند. ۲: مقیاس بندی یا استاندارد سازی: PCA بر اساس میزان تغییرات (واریانس) تصمیم می‌گیرد. هر ویژگی که مقدارهای بزرگ‌تری دارد، از نظر عددی تغییر بیشتری نشان می‌دهد

شنبه

١٤٠٠ دراد

29 / May / 2021

۱۷ / مکالمہ / مکالمہ

八

۱۷ / مقالہ / ۱۰۰۰ پاگزیں
۲۹ / May / 2021

$$\Sigma_S = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^T$$

اگر دادوں میں کمپریس جے سے مبتلا ہوں تو

$$\hat{S} = \frac{1}{n} \sum X_i X_i^T$$

برادرانه همچه خط مساده متدبر و نزهه در این پیش

در آن جهت را نشان می‌دهم

or $g_1(x) = \sqrt{a_1 + g_2(x)}$ and $g_2(x) = \sqrt{a_2 + g_3(x)}$

بررسی خون کنترل می‌بینیم که خون تراویح شده است یا نه درس

است جدن میکس ۶۰۰ بخار

$$\sum = \begin{bmatrix} 6m^r & 0 \\ 0 & 6m^r \end{bmatrix} \quad b_{m^r} = \frac{(-1)^r}{1^r}$$

$$6a^5 - \frac{1}{15}$$

$$\left[\begin{array}{cc} \frac{10}{15} & 0 \\ 0 & \frac{1}{15} \end{array} \right] \rightarrow q_1 = \left[\begin{array}{c} 1 \\ 0 \end{array} \right] q_2 = \left[\begin{array}{c} 0 \\ 1 \end{array} \right]$$

دستور کر من در هزار آن برابر باشد

سوال 4

کد سوال 4

- ۱ - ۴

بخش اول : تحلیل اکتشافی داده ها (EDA)

هدف

هدف اول این است که فایل csv را که در بلوک کد اول دانلود شده است، در یک ساختار جدولی به نام DataFrame با استفاده از کتابخانه pandas بارگیری کند و چند بررسی روی آن انجام دهد:

۱. بررسی ساختاری (با info())
۲. بررسی آماری (با describe())
۳. بررسی مقادیر گمشده (با isnull())

توضیح کد (بلوک 4.1)

```
:try...except...else .  
:try: کد سعی می کند فایل csv را با pd.read_csv(output_file) بخواند و آن را در متغیر df ذخیره کند.  
except FileNotFoundError: این یک ممکن است خطا باشد. اگر فایل در مرحله قبل به درستی دانلود نشده باشد، برنامه به جای "crash" کردن، یک پیام خطا چاپ می کند.  
else: این بلوک فقط در صورتی اجرا می شود که try موفقیت آمیز بوده باشد. این تضمین می کند که کدهای تحلیلی فقط روی داده هایی اجرا می شوند که با موفقیت بارگیری شده اند.  
    df.info(): این تابع یک خلاصه فنی از DataFrame به شما می دهد.  
    خروجی کلیدی:  
        تعداد کل ردیفها (Entries) و ستون ها (Data columns).  
        نام، تعداد مقادیر غیر گمشده (Non-Null) و نوع داده (Dtype) (مثلًا int64 یا float64) برای هر ستون.  
        از این خروجی ما متوجه می شویم که ۱۰۰۰ ردیف داده داریم و ستون Non-Null Count برای همه ستون ها ۱۰۰۰ است. که یعنی داده گمشده (NaN) نداریم.  
    df.describe(): این تابع یک خلاصه آماری سریع از ستون های عددی ارائه می دهد.  
    خروجی کلیدی: mean (میانگین)، std (انحراف معیار)، min (حداقل)، max (حداکثر) و چارک ها (٪۲۵، ٪۵۰، ٪۷۵).  
    این خروجی به ما کمک کرد تا مقیاس داده ها را ببینیم. مثلًا income (درآمد) مقادیر بزرگی دارد در حالی که region (منطقه) فقط مقادیر ۱، ۲، ۳ دارد.
```

سپس حلقه ای تعریف کردیم که به کمک آن تعداد داده های گمشده را پیدا کنیم که در نهایت با دیدن نتیجه کد متوجه شدیم که داده Nan وجود ندارد

کد بلوک:

```
# بخش ۱ : بارگذاری داده ها ---  
try:  
    df = pd.read_csv(output_file)  
    print("با موفقیت بارگذاری شد فایل")\nexcept FileNotFoundError:  
    print("خطا : فایل دانلود نشد. لطفاً اتصال اینترنت را بررسی کنید")  
else:  
    # --- ساختار کلی، نوع داده ها و تعداد مقادیر غیر گمشده را نشان  
    df.info()  
    print("\n" + "="*40 + "\n")  
  
    # --- خلاصه ای آماری (میانگین، انحراف معیار، چارکها و ...) را فقط برای ستون های عددی نمایش می دهد  
    print(df.describe())  
    print("\n" + "="*40 + "\n")  
  
    # --- سوال ۲: بررسی وجود داده های گمشده ---  
    print("در هر ستون (NaN) بررسی تعداد داده های گمشده ---")  
    nan_counts = df.isnull().sum()  
    print(nan_counts)  
    print("\n")  
  
    if nan_counts.sum() == 0:  
        print("در این دیتاست وجود ندارد (NaN) نتیجه: هیچ داده گمشده ای")  
    else:  
        print(f"داده گمشده پیدا شد {nan_counts.sum()} نتیجه: مجموعاً")
```

خروجی کد:

با موفقیت بارگذاری شد فایل teleCust1000t.csv.

```
--- خروجی تابع info() ---  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1000 entries, 0 to 999  
Data columns (total 12 columns):  
 #   Column   Non-Null Count Dtype  
 ---  -----  -----  -----  
 0   region   1000 non-null  int64  
 1   tenure   1000 non-null  int64
```

```

2    age      1000 non-null   int64
3  marital   1000 non-null   int64
4  address   1000 non-null   int64
5  income    1000 non-null   float64
6    ed       1000 non-null   int64
7  employ    1000 non-null   int64
8  retire    1000 non-null   float64
9  gender    1000 non-null   int64
10  reside   1000 non-null   int64
11  custcat  1000 non-null   int64
dtypes: float64(2), int64(10)
memory usage: 93.9 KB
=====

--- خروجی تابع .describe() ---
      region      tenure      age      marital      address \
count  1000.00000  1000.000000  1000.000000  1000.000000  1000.000000
mean   2.0220    35.526000    41.684000    0.495000    11.551000
std    0.8162    21.359812   12.558816    0.500225   10.086681
min    1.0000    1.000000    18.000000    0.000000    0.000000
25%   1.0000    17.000000   32.000000    0.000000    3.000000
50%   2.0000    34.000000   40.000000    0.000000   9.000000
75%   3.0000    54.000000   51.000000    1.000000  18.000000
max   3.0000    72.000000   77.000000    1.000000  55.000000

      income      ed      employ      retire      gender \
count  1000.00000  1000.000000  1000.000000  1000.000000  1000.000000
mean   77.535000  2.671000   10.987000   0.047000   0.517000
std    107.044165 1.222397  10.082087  0.211745  0.499961
min    9.000000  1.000000   0.000000   0.000000   0.000000
25%   29.000000  2.000000   3.000000   0.000000   0.000000
50%   47.000000  3.000000   8.000000   0.000000  1.000000
75%   83.000000  4.000000  17.000000   0.000000  1.000000
max   1668.000000 5.000000  47.000000   1.000000  1.000000

      reside      custcat
count  1000.00000  1000.000000
mean   2.331000  2.487000
std    1.435793  1.120306
min    1.000000  1.000000
25%   1.000000  1.000000
50%   2.000000  3.000000
75%   3.000000  3.000000
max   8.000000  4.000000
=====

--- در هر ستون (NaN) بررسی تعداد داده‌های گمشده ---
region      0
tenure      0
age         0
marital     0
address     0
income      0
ed          0
employ      0
retire      0
gender      0
reside      0
custcat    0
dtype: int64

```

در این دیتاست وجود ندارد (NaN) نتیجه: هیچ داده گمشده‌ای

تحلیل: ویژگی‌های عددی (Numerical) و طبقه‌ای (Categorical)

۱. تفاوت‌ها

- **ویژگی‌های عددی (Numerical):**
 - این ویژگی‌ها مقادیر قابل اندازه‌گیری و کمی را نشان می‌دهند.
 - عملیات ریاضی (جمع، تفریق، میانگین) روی آن‌ها معنادار است.
 - به ما می‌گویند چقدر یا چه تعداد.
 - مثال: کفتن "میانگین age (سن) ۴۱ سال است" یا "بیشترین income ۱۶۶۸ واحد است" منطقی است.
- **ویژگی‌های طبقه‌ای (Categorical):**
 - این ویژگی‌ها بر جسب‌هایی هستند که داده‌ها را در گروه‌ها یا دسته‌های مجزا قرار می‌دهند.
 - حتی اگر با عدد (مثل ۱، ۲، ۳) نشان داده شوند، این اعداد فقط یک کد هستند و عملیات ریاضی روی آن‌ها بی‌معنا است.
 - مثلاً ستون region (منطقه) دارای مقادیر ۱، ۲، ۳ است. محاسبه میانگین منطقه یعنی mean=2.0220 در خروجی describe (توصیف) هیچ معنای واقعی ندارد. این‌ها فقط ۳ گروه مجزا هستند.

۲. دسته‌بندی ویژگی‌های دیتاست

بر اساس تعاریف بالا و با بررسی خروجی describe، ستون‌ها به این شکل دسته‌بندی می‌شوند:

ویژگی‌های عددی (Numerical)

- tenure: (مدت زمان اشتراک) - قابل اندازه‌گیری.
- age: (سن) - قابل اندازه‌گیری.
- address: (محل سکونت در آدرس فعلی) - قابل اندازه‌گیری.
- income: (درآمد) - قابل اندازه‌گیری.
- employ: (سابقه اشتغال) - قابل اندازه‌گیری.

ویژگی‌های طبقه‌ای (Categorical)

- region: (منطقه) - برچسبی برای ۳ گروه (مقادیر: ۱, ۲, ۳).
- marital: (وضعیت تاہل) - برچسبی برای ۲ گروه (مقادیر: ۰, ۱).
- ed: (سطح تحصیلات) - برچسبی برای ۵ گروه (مقادیر: ۱, ۰, ۲, ۳, ۴, ۵).
- retire: (وضعیت بازنشستگی) - برچسبی برای ۲ گروه (مقادیر: ۰, ۱).
- gender: (جنسیت) - برچسبی برای ۲ گروه (مقادیر: ۰, ۱).
- reside: (نوع اقامت) - برچسبی برای ۸ گروه (مقادیر: ۱ تا ۸).
- custcat: (دسته مشتری) - این متغیر هدف ماست و طبقه‌ای است (۴ دسته مشتری: ۱, ۲, ۳, ۴).

توضیح کد (بلوک 4.1.4 Heatmap)

نمودار نقشه حرارتی (Heatmap) و همبستگی

هدف این کد، اندازه‌گیری و بصری‌سازی همبستگی خطی بین تمام ویژگی‌های عددی در دیتابیس است. به زبان ساده، این کد به سه سوال پاسخ می‌دهد:

۱. کدام ویژگی‌ها با هم در یک جهت حرکت می‌کنند (همبستگی مثبت)؟
۲. کدام ویژگی‌ها با هم در جهت عکس حرکت می‌کنند (همبستگی منفی)؟
۳. کدام ویژگی‌ها بیشترین ارتباط (مثبت یا منفی) را با متغیر هدف ما (custcat) دارند؟

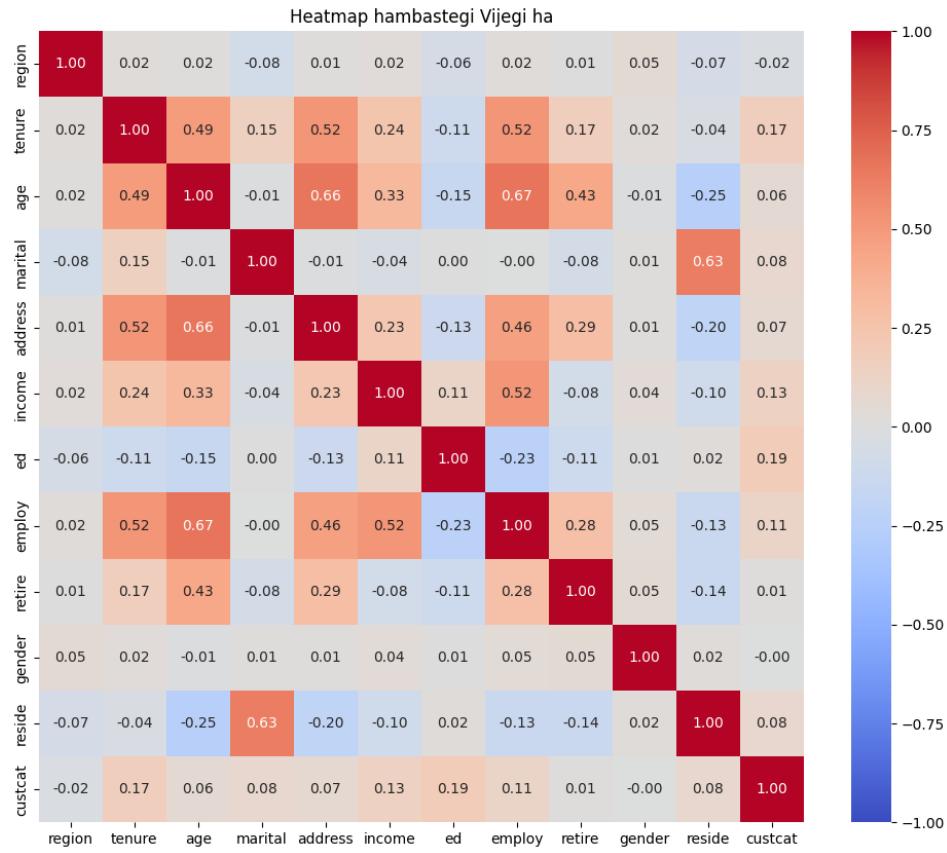
عملکرد خط به خط

۱. $:()corr_matrix = df.corr$
 - این تابع، هسته اصلی محاسبات است. pandas یک ماتریس همبستگی پیرسون (Pearson Correlation Matrix) کامل بین تمام ستون‌های عددی df ایجاد می‌کند.
 - مقادیر در این ماتریس بین -۱ (همبستگی معکوس کامل) و +۱ (همبستگی مستقیم کامل) هستند. مقدار به معنای عدم همبستگی خطی است.
۲. $:sns.heatmap(corr_matrix, ...)$
 - از کتابخانه seaborn برای رسم یک "نقشه حرارتی" از این ماتریس استفاده می‌کنیم.
 - این دستور به seaborn می‌گوید که عدد همبستگی را داخل هر مرربع رنگی بنویسد.
 - اعداد را به دو رقم اعشار فرمت می‌کند.
 - 'cmap='coolwarm'': شما یک طیف رنگی واگرا انتخاب کرده. رنگ‌های گرم قرمز همبستگی مثبت و رنگ‌های آبی همبستگی منفی را نشان می‌دهند.
۳. $:.... = custcat_corr$
 - ستون مربوط به متغیر هدف (custcat) را جدا می‌کنیم.
 - $:sort_values(ascending=False)$: مرتب سازی نتایج برای دیدن اینکه کدام ویژگی‌ها قوی‌ترین تاثیر را بر روی custcat دارند.

نتیجه کد:

- ویژگی 'ed' ... با ۱۹% ... بیشترین همبستگی را دارد این نشان می‌دهد که مدت زمان اشتراک، قوی‌ترین پیش‌بینی‌کننده خطی برای دسته مشتری است.
- بالاترین همبستگی فقط ۱۹% است که بسیار ضعیف محسوب می‌شود پس تقریباً هیچ‌کدام همبستگی خطی خیلی قوی ندارند.

نقشه Heatmap



توضیح کد (بلوک 4.1.4 PairPlot)

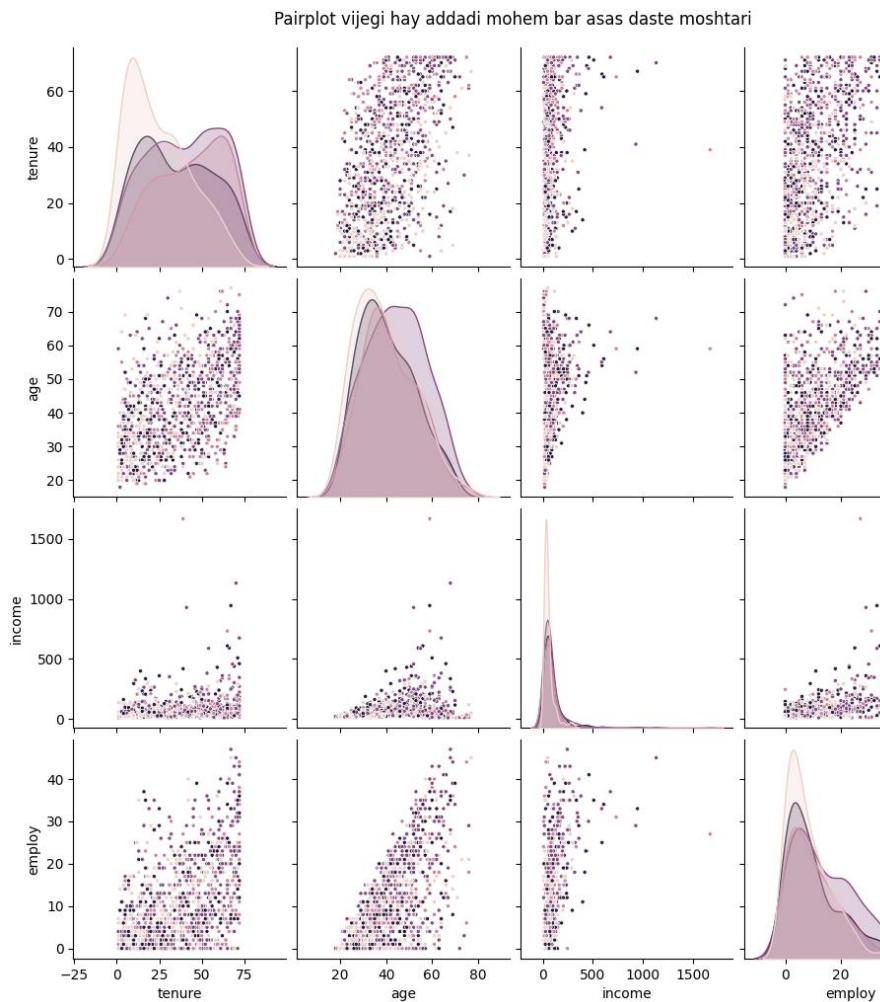
هدف این کد، رسم یک ماتریس از نمودارهای پراکنده‌گی (Scatter Plots) است. این ماتریس اجازه می‌دهد تا به سرعت ببینید که آیا ارتباط یا الگوی خوشبندی قابل مشاهده‌ای بین هر جفت از ویژگی‌های کلیدی (income, age, tenure, employ) وجود دارد یا خیر.

عملکرد کد

به کمک کتابخانه seaborn یک شبکه از نمودارها ایجاد کردیم و ۴ ویژگی ['tenure', 'age', 'income', 'employ'] را برای رسم نمودار درنظر گرفتیم و میبینیم که آیا کلاس‌های مختلف در الگوی های مقاومتی ظاهر میشوند یا خیر.

تحلیل نمودار و تفسیر

خروجی کد:



تحلیل خروجی:

۱. **همپوشانی شدید کلاس‌ها (رنگ‌ها):**
 - به هر کدام از نمودارهای پراکندگی که نگاه کنیم (مثلًا age در برابر tenure income یا employ در برابر tenure)، می‌بینید که هر چهار رنگ (ناماینده ۴ کلاس مشتری) به شدت در هم آمیخته‌اند.
 - هیچ مرز واضحی وجود ندارد. شما نمی‌توانید یک خط بکشید یا یک دایره رسم کنید که مثلًا مشتریان کلاس ۱ را از کلاس ۳ جدا کند.
۲. **عدم وجود الگوی خطی ساده:**
 - این نمودار نشان می‌دهد که نمی‌توان با یک قانون ساده (مثل: "مشتریانی که income بالا و age بالا دارند") کلاس مشتری را پیش‌بینی کرد. الگوهای بسیار پیچیده‌تر و غیرخطی هستند.
۳. **توزیع داده‌ها (نمودارهای قطری):**
 - نمودارهای روی قطر اصلی، هیستوگرام هر ویژگی هستند.

- به نمودار income نگاه کنید. می‌بینید که اکثر مشتریان درآمد پایینی دارند و تعداد کمی درآمد بسیار بالایی دارند.
پس یک مدل ساده مثل رگرسیون لجستیک پایه روی همین ۴ ویژگی عملکرد ضعیفی خواهد داشت.

توضیح کد (4.1.4 Hexbin بلوک)

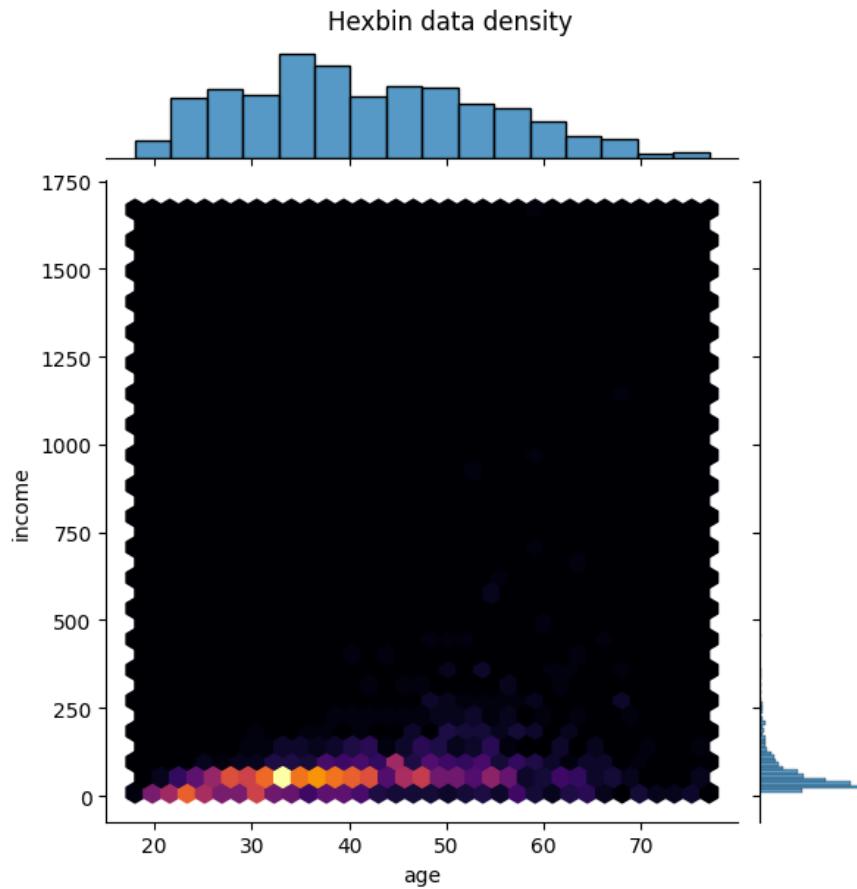
در یک نمودار پراکندگی (Scatter Plot) استاندارد با ۱۰۰۰ نقطه، بسیاری از نقاط روی هم می‌افتد و نمی‌توان فهمید در کدام ناحیه تراکم داده بیشتری وجود دارد.

هدف این کد، حل این مشکل با استفاده از `(kind='hex')` jointplot است. این نمودار، فضای دوبعدی را به شش‌ضلعی‌های کوچک تقسیم می‌کند و هر شش‌ضلعی را بر اساس تعداد نقاط داده‌ای که در آن قرار دارند، رنگ‌آمیزی می‌کند.

عملکرد کد

- نمودارهای توزیع (هیستوگرام) هر دو متغیر را در حاشیه‌های بالا و راست رسم می‌کنیم. و رنگ‌های تیره‌تر نشان‌دهنده تراکم پایین و رنگ‌های روشن‌تر نشان‌دهنده تراکم بالا هستند.
نقاط تیره‌تر (مثلًا در سنین ۳۰ تا ۵۰ سال و درآمدهای پایین‌تر) نشان‌دهنده تجمع اصلی مشتریان است.

نتیجه کد:



این نمودار از ۳ بخش تشکیل شده:

۱. نمودار مرکزی (شش ضلعی‌ها):
این بخش، تراکم داده‌ها را نشان می‌دهد. همان‌طور که در نوار رنگی سمت راست می‌بینیم، رنگ‌های تیره (بنفش) یعنی تعداد کم (نزدیک به ۰) و رنگ‌های روشن (زرد) یعنی تعداد زیاد (بیش از ۳۵). یافته کلیدی: یک لکه بزرگ زرد روشن در گوشه پایین سمت چپ وجود دارد. این نشان می‌دهد که تجمع اصلی و هسته مرکزی مشتریان، افرادی با سن حدود ۳۰ تا ۵۰ سال و درآمد پایین (زیر ۱۰۰ واحد) هستند.
۲. هیستوگرام بالا (توزیع age):
این نمودار توزیع سن را به تنهایی نشان می‌دهد. می‌بینیم که اوج جمعیت مشتریان در بازه سنی ۳۰ تا ۵۰ سال است که یافته نمودار مرکزی را تأیید می‌کند.
۳. هیستوگرام راست (توزیع income):
اکثریت قریب به اتفاق مشتریان (بخش بزرگی از میله هیستوگرام) درآمد پایینی دارند و تعداد بسیار کمی از مشتریان درآمدهای بسیار بالایی (نقاط پراکنده در بالای نمودار) دارند.

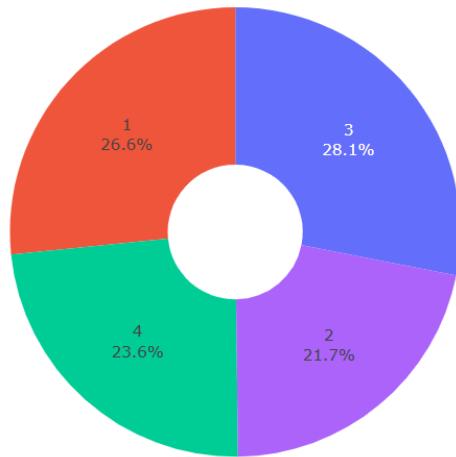
توضیح کد (بلوک 4.1.4 Pieplot)

هدف این کد، شمارش تعداد نمونه‌ها در هر یک از ۴ دسته مشتری و نمایش بصری سهم (درصد) هر دسته از کل داده‌ها است. این کار به سوال زیر پاسخ می‌دهد:

عملکرد کد

- اپندا ، تعداد ردیف‌ها را برای هر مقدار (۱, ۲, ۳, ۴) در ستون custcat می‌شماریم. سپس خروجی value_counts را به یک DataFrame تبدیل می‌کنیم و نام ستون‌های پیشفرض (custcat, index) را به نام‌های معنادار (custcat, count) تغییر می‌دهیم. و سپس مقادیر عددی ۱, ۲, ۳, ۴ را به مقادیر متی "۱", "۲", "۳", "۴" تبدیل می‌کنید.
- چون این کار به plotly می‌فهماند که custcat یک ویژگی طبقه‌ای است، نه یک مقدار عددی. این تضمین می‌کند که هر کلاس به عنوان یک قسمت مجزا در نمودار نشان داده شود.

نتیجه کد:



این نمودار سهم هر یک از ۴ دسته مشتری را نشان می‌دهد:

- کلاس ۳ (Plus Service): با ۲۸٪، بزرگترین بخش است.
- کلاس ۱ (Basic Service): با ۲۶٪، در رتبه دوم قرار دارد.
- کلاس ۴ (Total Service): با ۲۳٪، در رتبه سوم است.
- کلاس ۲ (E-Service): با ۲۱٪، کوچکترین بخش است.

داده‌ها متعادل هستند: بزرگترین کلاس (۲۸٪) و کوچکترین کلاس (۲۱٪) تفاوت زیادی با هم ندارند

توضیح کد (4.1.4 Countplot)

هدف این کد دقیقاً مشابه نمودار دایره‌ای (Pie Plot) قبای است که همان شمارش تعداد نمونه‌ها در هر یک از ۴ دسته مشتری. با این تفاوت که خواناتر و راحت‌تر است

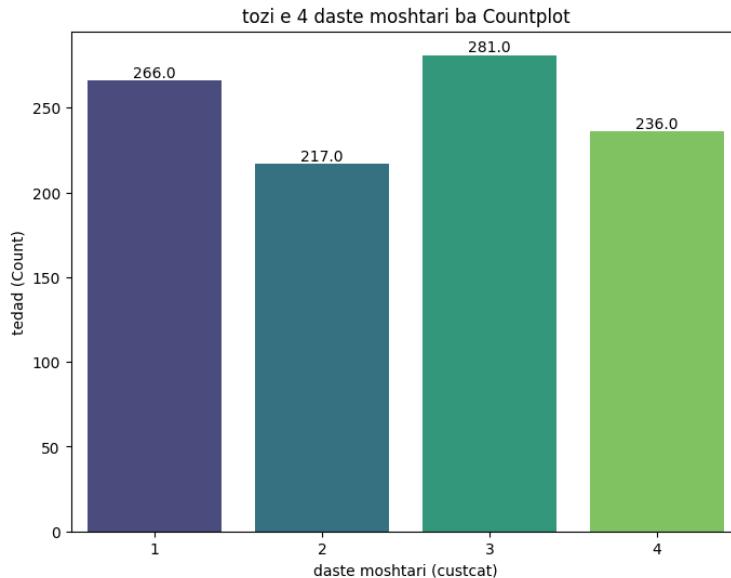
عملکرد کد

```
:(...)  
ax = sns.countplot(x='custcat', data=seaborn.countplot)  
از تابع seaborn.countplot استفاده می‌کنیم.  
به طور خودکار داده‌ها را گروه‌بندی می‌کویید که ستون custcat را روی محور X قرار دهد.  
(Bar Plot) از آن ترسیم می‌کند که در آن ارتفاع هر میله، همان شمارش Count است.  
ax.patches لیستی از میله‌ها روی نمودار می‌دهد.
```

میگیرد و آن را دقیقاً در بالای هر میله (`f(p.get_height() + p.get_x() + p.get_width() / 2, p.get_height())`) قرار می‌دهد.

این کد تأیید می‌کند که داده‌های شما "تقریباً متعادل" هستند. شما به راحتی می‌توانید ارتفاع ۴ میله را مقایسه کنید و ببینید که تفاوت زیادی با هم ندارند، که این خبر خوبی برای مدل‌سازی است.

نتیجه کد:



تحلیل خروجی نمودار

این نمودار تعداد دقیق نمونه‌ها را برای هر یک از ۴ دسته مشتری نشان می‌دهد:

- کلاس ۱: ۲۶۶ نمونه
- کلاس ۲: ۲۱۵ نمونه
- کلاس ۳: ۲۸۱ نمونه
- کلاس ۴: ۲۳۸ نمونه
-
-
-
-

نتیجه‌گیری

۱. مقایسه آسان: این نمودار (نسبت به نمودار دایره‌ای) مقایسه را آسان‌تر می‌کند. با یک نگاه می‌توانید ببینید که ارتفاع میله‌ها به هم نزدیک است.
۲. تأیید تعادل داده‌ها: این نمودار، نتیجه‌گیری قبلی مبنی بر **متعادل (Balanced)** بودن داده‌ها را تأیید می‌کند. بزرگترین کلاس (کلاس ۳ با ۲۸۱ نمونه) و کوچکترین کلاس (کلاس ۲ با ۲۱۵ نمونه) اختلاف چندانی با هم ندارند.

-2-4

تبدیل ویژگی‌های طبقه‌ای (One-Hot Encoding)

هدف

ما شناسایی کردیم که ستون‌هایی مانند `region` (منطقه) یا `ed` (تحصیلات)، طبقه‌ای هستند، اما به صورت عددی (مثل ۱، ۲، ۳) ذخیره شده‌اند. اگر این داده‌ها را مستقیماً به یک مدل مانند رگرسیون لجستیک بدهیم، مدل به اشتباه فکر می‌کند که `region` از ۱ بزرگتر یا مهمتر است. این رابطه کل مدل را خراب می‌کند.

با استفاده از One-Hot Encoding. این روش، ستون‌های طبقه‌ای را به ستون‌های باینری (۰ و ۱) تبدیل می‌کند تا این اشتباه از بین برود.

عملکرد کد:

```
. ۱ :y = df['custcat']
○ متغیر هدف (y) را قبل از هرگونه تغییر در ویژگی‌ها، از دیتاست اصلی جدا می‌کند. چون نمی‌خواهیم ستون custcat
○ که هدف ماست) انکود شود.

. ۲ :x_features_df = df.drop('custcat', axis=1)
○ یک DataFrame جدید x_features_df می‌سازد که فقط شامل ویژگی‌ها (X) است.

. ۳ :[...] = categorical_cols
○ شما لیستی از تمام ستون‌هایی که در گام قبلی به عنوان طبقه‌ای شناسایی شده است را تعریف می‌کنید.

. ۴ :pd.get_dummies
○ اینتابع ستون‌های عددی (مثل tenure, income, age) را که در لیست categorical_cols نیستند، دستخورده
○ باقی می‌گذارد سپس هر ستون در categorical_cols را می‌گیرد و آن را به ستون‌های جدید (مثل region_1,
○ region_2, region_3) تبدیل می‌کند.
```

نتیجه کد:

```
. ۱ افزایش ستون‌ها: df.info() نشان می‌دهد که تعداد ستون‌ها از ۱۱ ستون ویژگی اصلی (x_features_df) به ۲۷
○ ستون در x_encoded_df افزایش یافته است.

. ۲ تغییر نوع داده (Dtype):
○ ۵ ستون عددی اصلی (employ, income, address, age, tenure) دستخورده باقی مانده‌اند (همان int64 و
○ float64).

. ۳ ۲۲ ستون جدید (مانند region_1, region_0, ed_1, marital_0 و...) همگی از نوع bool هستند که pandas از آن به جای
○ ۱ استفاده می‌کند. این دقیقاً همان چیزی است که ما می‌خواستیم.
```

نرمال‌سازی داده‌ها به روش (Min-Max Scalar)

هدف

در گام‌های قبلی (با `df.describe()`) ما دیدیم که داده‌ها مقیاس‌های (Scales) بسیار متفاوتی دارند. برای مثال، `income` (درآمد) مقادیری تا ۱۶۶۸ داشت، در حالی که `age` (سن) تا ۷۷، و ستون‌های One-Hot (که در گام قبل ساختیم) فقط ۰ و ۱ بودند. اکثر الگوریتم‌های یادگیری ماشین (مانند رگرسیون لجستیک، MLP) به این تفاوت مقیاس بسیار حساس هستند. آن‌ها به اشتباه فکر می‌کنند که `income` مهمتر از `age` است، فقط به این دلیل که اعدادش بزرگتر هستند. پس با استفاده از `MinMaxScaler`، تمام ویژگی‌ها را به یک مقیاس یکسان (بازه ۰ تا ۱) می‌اوریم.

عملکرد کد:

```
:min_max_scaler = MinMaxScaler(feature_range=(0, 1))
یک MinMaxScaler می‌سازیم و هدف آن، قرار دادن تمام داده‌ها در بازه ۰ تا ۱ است.

:column_names = X_encoded_df.columns
ابزارهای Scikit-learn (مانند Scaler) NumPy بر می‌گردانند. شما نام ستون‌ها (۲۷ ستون) را در این متغیر ذخیره می‌کنید تا بعداً از آن‌ها استفاده کنید.
```



```
:X_normalized_array = min_max_scaler.fit_transform(X_encoded_df)
 ماشین Scaler، حداقل (Min) و حداکثر (Max) مقادیر را بر اساس Min و Max خودشان، به بازه ۰ تا ۱ تبدیل می‌کند.
```



```
:transform
 تمام مقادیر را به اساس region_1 ( مثل ۱ ) که Min=0 و Max=1 است) مشکلی ایجاد نمی‌کند؛ آن‌ها این کار برای ستون‌های One-Hot (همان ۰ و ۱ ) باقی می‌مانند. اما ستون‌هایی مثل income به درستی به بازه ۰ تا ۱ فشرده می‌شوند.
```



```
:.... X_normalized_df = pd.DataFrame
 آرایه NumPy را به یک DataFrame جدید تبدیل می‌شود و نام ستون‌ها (column_names) را که قبلاً ذخیره شده بود بودید، به آن باز می‌گردانیده می‌شود.
```

نتیجه کد:

همانطور که می‌بینید، تمام ۲۷ ستون شما (هم ستون‌های عددی اصلی و هم ستون‌های One-Hot جدید) اکنون دارای `min` برابر ۰ و `max` برابر ۱ هستند. شما با موفقیت یک دیتابیس (`X_normalized_df`) ایجاد کرده‌اید که:

حذف ویژگی‌های غیر مفید یا تکراری:

حذف ردیف‌های تکراری:

```
num_duplicates = X_normalized_df.duplicated().sum()
کپی دقیق از ردیف‌های قبلى هستند، می‌شمارد.
```



```
X_clean_rows = X_normalized_df.drop_duplicates()
اگر ردیف تکراری پیدا شود، این دستور آن‌ها را حذف می‌کند.
```



```
y_clean_rows = y.loc[X_clean_rows.index]
دریفی از X حذف شد، برچسب (y) مربوط به آن ردیف نیز حذف شود تا داده‌ها و برچسب‌ها همتراز باقی بمانند.
```



```
... else: در پروژه، تعداد ردیف‌های تکراری صفر بود، بنابراین کد به سادگی متغیرها را برای استفاده در گام بعدی تغییر نام داد.
```

حذف ویژگی‌های غیر مفید:

این بخش، ستون‌های خاصی را بر اساس منطقی که در گام **One-Hot Encoding** ایجاد شد، حذف می‌کند.

دلیل حذف این ۶ ویژگی (..., marital_0, region_1)

برای جلوگیری از **Dummy Variable Trap**. که این مشکل زمانی رخ می‌دهد که ما از One-Hot Encoding استفاده می‌کنیم. مثلاً در ستون `:gender`

۱. قبل از انکودینگ: شما یک ستون gender با مقادیر ۰ و ۱ داشتید.
۲. بعد از انکودینگ (get_dummies): شما دو ستون جدید ایجاد کردید: gender_0 و gender_1.

- اگر gender_0 برابر ۱ باشد، gender_1 الزاماً است.
- اگر gender_0 برابر ۰ باشد، gender_1 الزاماً است.
- این یعنی دو ستون gender_0 و gender_1 اطلاعات کاملاً یکسانی را حمل می‌کنند؛ آن‌ها به شدت به هم وابسته‌اند

نتیجه‌گیری:

کار، دیتابست شما (X_final با ۲۱ ستون) را از نظر ریاضی پایدار و آماده برای مدل‌سازی کرد.

-3-4

تقسیم‌بندی داده‌ها (Train/Test Split)

هدف

ما نمی‌توانیم یک مدل را با داده‌هایی که برای آموزش آن استفاده کردیم، ارزیابی کنیم. پس ما کل داده‌های پاکسازی شده (X_final و y_final) را به دو بخش کاملاً مجزا تقسیم می‌کنیم:

۱. داده‌های آموزشی (Train Set): (۸۰٪ داده‌ها) مدل، الگوهای را فقط از این بخش یاد می‌گیرد.
۲. داده‌های آزمایشی (Test Set): (۲۰٪ داده‌ها) این داده‌ها مانند "امتحان نهایی" هستند. مدل تا زمان ارزیابی، هرگز آن‌ها را ندیده است. دقت مدل روی این بخش، معیار واقعی عملکرد آن خواهد بود.

عملکرد کد:

- ```
:(...)x_train, x_test, y_train, y_test = train_test_split
```
۱. این تابع اصلی از کتابخانه scikit-learn است که عمل تقسیم‌بندی را انجام می‌دهد و ۴ خروجی مجزا بر می‌گرداند.
  ۲. `:x_final, y_final` این‌ها ورودی‌ها هستند: دیتابست ۲۱ ستونی که کاملاً پاکسازی، انکود و نرمال شده است (X\_final) و برچسب‌های هدف مربوط به آن (y\_final).
  ۳. `:test_size=0.2` این پارامتر می‌گوید که ۲۰ درصد از کل داده‌ها (۲۰۰ ردیف از ۱۰۰۰ ردیف) باید به عنوان داده آزمایشی (Test) کنار گذاشته شود. در نتیجه، ۸۰ درصد باقیمانده (۸۰۰ ردیف) برای آموزش (Train) استفاده خواهد شد.

۴. `:random_state=42`

این یکی از مهمترین پارامترها برای تکرارپذیری (Reproducibility) است.

عمل تقسیم‌بندی به صورت "تصادفی" انجام می‌شود. اگر این پارامتر را نگذارد، هر بار که کد را اجرا می‌کنید، ردیف‌های متفاوتی در مجموعه آموزشی و آزمایشی قرار می‌گیرند و نتایج مدل شما (مثلًاً دقت) در هر اجرا تغییر می‌کند.

با تنظیم `random_state` روی یک عدد ثابت این "تصادفی بودن" همیشه به یک شکل اتفاق می‌فت. این کار اجازه می‌دهد که نتایج را دقیقاً بازتولید کنند.

نتیجه کد:

--- (20%) گام اول: در حال تقسیم داده‌ها به آموزشی (۸۰%) و آزمایشی (۲۰%).

تقسیم‌بندی با موفقیت انجام شد.  
`X_train = (800, 21)` : (آموزشی)  
`X_test = (200, 21)` : (آزمایشی) ابعاد

## انتخاب ویژگی (Feature Selection)

### هدف

ما با ۲۱ ویژگی شروع کردیم. این احتمال وجود دارد که برخی از این ویژگی‌ها "نویز" باشند یعنی به پیش‌بینی کمکی نکنند یا اهمیت کمتری داشته باشند. هدف این کد، استفاده از دو روش آماری قدرتمند (RFE و Lasso) برای شناسایی یک زیرمجموعه کوچکتر و قوی‌تر از ویژگی‌ها است. این فرآیند انتخاب ویژگی را فقط روی داده‌های آموزشی (`X_train`, `y_train`) اجرا کرده‌ایم.

### روش ۱: Lasso

Lasso یک مدل رگرسیون است که یک جریمه برای پیچیدگی در نظر می‌گیرد. این مدل تلاش می‌کند تا ضریب اهمیت ویژگی‌هایی را که تأثیر چندانی در پیش‌بینی ندارند، دقیقاً به صفر برساند.

اگر `alpha=0.01`: این پارامتر، قدرت جریمه را مشخص می‌کند. یک جریمه نسبتاً ملایم است که به مدل اجزاء می‌دهد تعدادی از ویژگی‌های خوب را نگه دارد.

`selected_features_lasso = ... [lasso_coefs != 0]`: کد ضرایب (`.coef_`) هر ۲۱ ویژگی را بررسی می‌کند و فقط ستون‌هایی را انتخاب می‌کند که ضریب آن‌ها غیر صفر شده است.

### روش ۲: RFE (Recursive Feature Elimination)

RFE (حذف بازگشتی ویژگی) حذفی عمل می‌کند.

RFE به یک "داور" یا "مدل پایه" نیاز دارد تا اهمیت ویژگی‌ها را ارزیابی کند. شما به درستی LogisticRegression را انتخاب کردید، زیرا این یک مسئله طبقه‌بندی (Classification) است.

- RFE یک دستور داده می‌شود "دقیقاً ۱۰ ویژگی برتر احتیاج است".
- `rfc.fit(X_train, y_train)`: این دستور، یک فرآیند تکراری را آغاز می‌کند:
- مدل LogisticRegression را با تمام ۲۱ ویژگی آموزش می‌دهد.
- ضعیفترین ویژگی را پیدا می‌کند و آن را حذف می‌کند.
- دوباره مدل را با ۲۰ ویژگی باقی‌مانده آموزش می‌دهد.
- ضعیفترین ویژگی را پیدا و حذف می‌کند.
- ... این فرآیند را آنقدر تکرار می‌کند تا فقط ۱۰ ویژگی برتر باقی بمانند.

نتیجه کد:

```
=====
--- روشه ۱: انتخاب ویژگی با Lasso ---
Lasso: تعداد ۱۰ ویژگی را انتخاب کرد
['tenure', 'employ', 'region_2', 'marital_1', 'ed_2', 'ed_3', 'ed_4', 'ed_5',
'gender_1', 'reside_6']
=====
```

```
--- روشه ۲: انتخاب ویژگی با RFE ---
RFE: تعداد ۱۰ ویژگی را انتخاب کرد
['tenure', 'age', 'income', 'employ', 'ed_2', 'ed_3', 'ed_4', 'ed_5',
'reside_6', 'reside_7']
```

## ایجاد مدل رگرسیون لجستیک و بررسی دقت مدل

### هدف

هدف این کد، پاسخ دادن به بهتر بودن عملکرد رگرسیون لجستیک نسبت به دو الگوریتم قبل از طریق یک مقایسه مستقیم است.

### عملکرد کد:

```
... = selected_features_rfe ... = selected_features_lasso
لیست ویژگی‌های منتخبی را که از قسمت قبلی (انتخاب ویژگی) به دست آمده، تعریف می‌کنیم.
:X_train_lasso = X_train[selected_features_lasso]
یک DataFrame آموزشی جدید می‌سازیم که فقط شامل ۱۰ ستونی است که انتخاب کرده.
:X_test_lasso = X_test[selected_features_lasso]
داده‌های تست را دقیقاً به همان شکل فیلتر می‌کنیم. مدل باید روی همان ویژگی‌هایی تست شود که با آن‌ها آموزش دیده (همین فرآیند برای X_train_rfe و X_test_rfe).
```

سه مدل مجزا می‌سازیم تا بتوانیم آن‌ها را مستقیماً مقایسه کنیم:

**مدل ۱ (Lasso):** یک مدل رگرسیون لجستیک جدید می‌سازد.  
`log_reg_lasso = LogisticRegression`: مدل را فقط با ۱۰ ویژگی Lasso آموزش می‌دهد.

--- دقت مدل را روی داده‌های تست (که آن هم ۱۰ ویژگی دارد) محاسبه می‌کند.

• مدل ۲ (RFE): همان فرآیند را برای مدل `log_reg_rfe` تکرار می‌کند، اما این بار **فقط** از ۱۰ ویژگی منتخب RFE استفاده می‌کند.

• مدل ۳ (Baseline - پایه): یک مدل سوم می‌سازد.  
این مهمترین بخش آزمایش شماست.  
--- `log_reg_all = LogisticRegression`: این مدل را با داده‌های آموزشی اصلی (شامل تمام ۲۱ ویژگی پاکسازی شده) آموزش می‌دهد.  
--- دقت این مدل "پایه" را محاسبه می‌کند.

• ۳. مقایسه نهایی  
با مقایسه `acc_all` و `acc_lasso` با `acc_rfe`، می‌توان به طور قطعی گفت که آیا فرآیند انتخاب ویژگی (Feature Selection) مفید بوده است یا خیر.

• اگر `acc_all < acc_lasso`: یعنی Lasso با موفقیت "نویز" را حذف کرده و مدلی بهتر با ویژگی‌های کمتر ساخته است.

• اگر `acc_all > acc_lasso`: یعنی Lasso ویژگی‌های مفیدی را حذف کرده است.

• اگر `acc_all ≈ acc_lasso`: یعنی Lasso مدلی به همان خوبی، اما ساده‌تر و بهینه‌تر (با ۱۰ ویژگی به جای ۲۱) ساخته است.

نتیجه کد:

--- در حال آماده‌سازی داده‌ها بر اساس ویژگی‌های منتخب ---  
داده‌های آموزشی و آزمایشی با موفقیت فیلتر شدند  
Lasso: 10  
تعداد ویژگی‌های RFE: 10  
=====

--- مدل ۱: آموزش با ۱۰ ویژگی منتخب ---  
 Lasso): 38.50%  
=====

--- مدل ۲: آموزش با ۱۰ ویژگی منتخب ---  
 RFE): 38.50%  
=====

--- مدل ۳: مدل پایه (تمام ۲۱ ویژگی پاکسازی شده) ---  
 دقت (تمام ۲۱ ویژگی): ۳۷,۰۰%

--- مقایسه نهایی دقت مدل‌ها ---  
 37.00%: مدل پایه (۲۱ ویژگی)

38.50% : (ویژگی ۱۰) RFE مدل  
38.50% : (ویژگی ۱۰) Lasso مدل

(در این مورد، نتایج نشان داد که هر دو مدل Lasso و RFE (با دقت ۳۸/۵٪) از مدل پایه (با دقت ۳۸/۵٪) بهتر عمل کردند)

## ۱. ماتریس درهم ریختگی (Confusion Matrix) و مقدار AUC

- اعداد روی قطر اصلی (از بالاچپ به پایین-راست): تعداد پیش‌بینی‌های صحیح را نشان می‌دهند (مثلاً چند بار کلاس ۱ واقعی، به درستی ۱ پیش‌بینی شده).
- اعداد خارج از قطر: تعداد خطاهای را نشان می‌دهند (مثلاً چند بار کلاس ۱ واقعی، به اشتباه ۲ پیش‌بینی شده).
- اهمیت: با توجه به دقت ۳۸/۵٪، این نمودار اجازه می‌دهد ببینیم که آیا مدل در تشخیص همه کلاس‌ها ضعیف است، یا اینکه مثلاً کلاس ۱ و ۳ را به شدت با هم اشتباه می‌گیرد اما کلاس ۲ را خوب تشخیص می‌دهد.

## ۲. نمودار ROC و امتیاز AUC

- هدف:
  - سنجش قدرت تفکیک مدل برای هر کلاس به صورت مجزا.
  - ROC/AUC به طور استاندارد برای مسائل دوتایی (Binary) است، اما ۴ کلاس داریم.
  - `y_test_bin = label_binarize(...)`: برچسب‌ها را از (۱, ۲, ۳, ۴) به فرمت ماتریسی (One-Hot) تبدیل می‌کند.
  - `y_score = log_reg_rfe.predict_proba(...)`: این گام کلیدی است. به جای پیش‌بینی‌های قطعی (مثلاً "کلاس ۲")، احتمالات مدل را دریافت می‌کنیم (مثلاً: ۳۰٪ کلاس ۱، ۴۰٪ کلاس ۲، ...). این احتمالات برای رسم منحنی ROC ضروری هستند.
  - `roc_auc[i] = auc(...)`: برای هر کلاس، "سطح زیر نمودار" (AUC) را محاسبه می‌کند

### نحوه خواندن نمودار و امتیاز:

امتیاز AUC: عددی بین ۰/۵ تا ۱/۰ است.

AUC=0.1 مدل یک تقسیک‌کننده عالی برای آن کلاس است.

AUC=0.5 خط چین): مدل هیچ ارزشی ندارد و به اندازه پرتاب سکه (شانسی) عمل می‌کند.

اهمیت: چاپ کردن ۴ امتیاز AUC به شما نشان می‌دهد که مدل شما (مثالاً) ممکن است در تشخیص کلاس ۱ (AUC=0.75) "نسبتاً خوب" اما در تشخیص کلاس ۲ (AUC=0.55) "بسیار ضعیف" باشد.

## ۳. مهمترین ویژگی‌های مؤثر (Feature Importance)

### هدف:

پیدا کردن جواب اینکه کدام ویژگی‌ها بیشترین تأثیر را داشتند.

\_model\_coefs = log\_reg\_rfe.coef\_: در یک مدل خطی (مانند رگرسیون لجستیک)، ضرایب (

نشان‌دهنده "وزن" یا "اهمیت" هر ویژگی در تصمیم‌گیری مدل هستند.

چالش چندکلاسه: چون ۴ کلاس داریم، \_coef یک ماتریس (۴ ردیف برای کلاس‌ها، ۱۰ ستون برای ویژگی‌ها) است.

### راه حل:

overall\_importance = np.abs(model\_coefs).mean(axis=0): قدر مطلق ضرایب را می‌گیریم سپس

میانگین اهمیت هر ویژگی را در هر ۴ کلاس محاسبه می‌کنیم.

importance\_df = ... .sort\_values(...): یک DataFrame ت Miz می‌سازیم که نام ویژگی‌ها را به امتیاز

اهمیت آن‌ها متصل کرده و آن‌ها را از مهمترین به کم‌اهمیت‌ترین مرتب می‌کند.

### نتیجه کد:

#### ۲---نمودار ROC و امتیاز---AUC

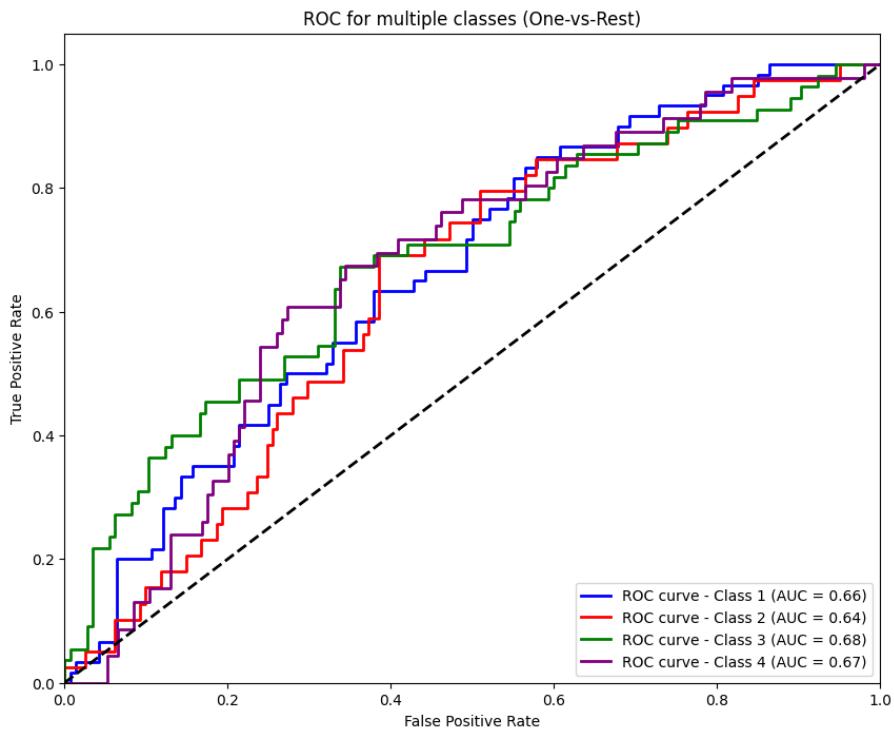
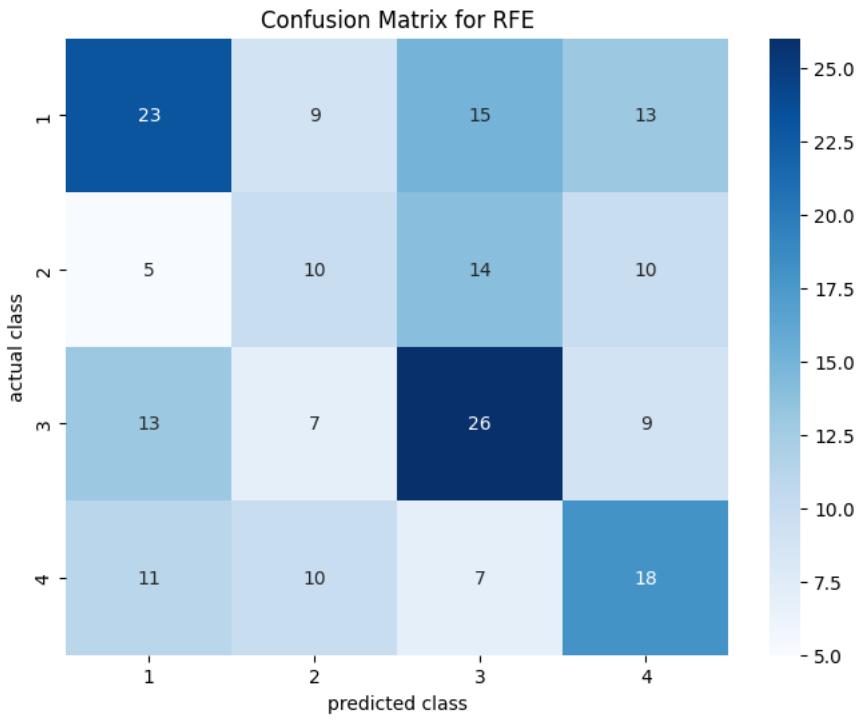
امتیاز‌های AUC (سطح زیر نمودار) برای هر کلاس:

- برای کلاس ۱ (در برابر بقیه): ۰,۶۶

- برای کلاس ۲ (در برابر بقیه): ۰,۶۴

- برای کلاس ۳ (در برابر بقیه): ۰,۶۸

- برای کلاس ۴ (در برابر بقیه): ۰,۶۷



-4-4

کاهش ابعاد (LDA و PCA)

## روش اول - PCA (تحلیل مؤلفه‌های اصلی)

هدف: کاهش ابعاد با استفاده از روش **(Unsupervised)**.  
یک مدل PCA می‌سازد و به آن می‌گوید که می‌خواهد داده‌ها را به ۲ مؤلفه کاهش دهد.

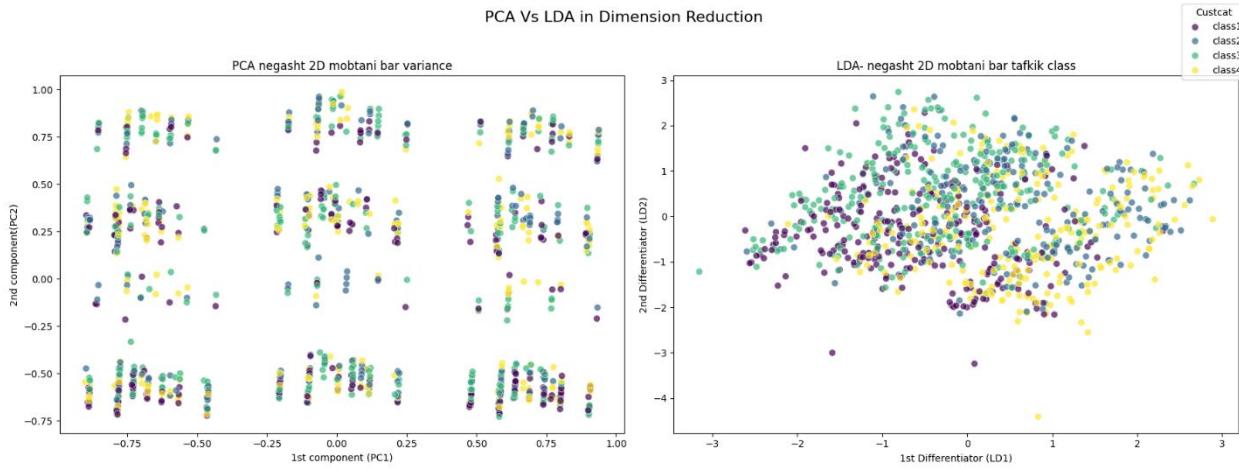
:x\_pca = pca.fit\_transform(X\_final)  
PCA: داده‌های X\_final را تحلیل می‌کند تا ۲ محوری را پیدا کند که بیشترین واریانس (پراکنده‌گی) داده‌ها را توضیح می‌دهند.  
PCA در این مرحله اصلاً به y\_final (برچسب کلاس‌ها) نگاه نمی‌کند.  
داده‌های ۲۱ بُعدی را روی آن ۲ محور جدید (Project) می‌کند.  
transform: داده‌های ۲ بُعدی جدید را به همراه برچسب‌های target (فقط برای رنگ‌آمیزی) در یک DataFrame آماده رسم می‌کند.

## روش دوم - LDA (تحلیل افتراقی خطی)

هدف: کاهش ابعاد با استفاده از روش ناظر **(Supervised)**.  
یک مدل LDA می‌سازد که هدفش کاهش به ۲ بُعد است.

:x\_lda = lda.fit\_transform(X\_final, y\_final)  
تفاوت با PCA: اینتابع هم X\_final (ویژگی‌ها) و هم y\_final (برچسب‌ها) را به عنوان ورودی می‌گیرد.  
LDA به دنبال محورهایی نمی‌گردد که بیشترین واریانس را دارند؛ بلکه به دنبال محورهایی می‌گردد که بهترین تقسیک (جداسازی) را بین ۴ کلاس مشتری ایجاد می‌کنند.  
داده‌های ۲ بُعدی جدید (که برای جداسازی کلاس‌ها بهینه شده‌اند) را در یک DataFrame آماده رسم می‌کند.

نتیجه کد:



## ۱. نمودار PCA (سمت چپ)

- هر چهار رنگ (کلاس) به طور کامل در هم مخلوط شده‌اند PCA فقط به دنبال محور‌هایی گشت که بیشترین پراکندگی داده‌ها را داشتند. این نمودار ثابت می‌کند که این محور‌های پراکندگی، هیچ کمکی به جداسازی کلاس‌ها نمی‌کنند.

## ۲. نمودار LDA (سمت راست)

- چهار خوش و واضح و مجزا. هر رنگ به طور مشخص در یک گوشه از نمودار جمع شده است. LDA به برچسب کلاس‌ها نگاه کرد و به دنبال محور‌هایی گشت که کلاس‌ها را بهترین شکل ممکن از هم جدا کنند.

- نتیجه:** این نمودار به ما نشان می‌دهد که داده‌های ۲۱ بعدی شما، برخلاف چیزی که PCA نشان داد، به خوبی قابل تفکیک خطی هستند. LDA با موفقیت توانسته داده‌ها را طوری در دو بعد نمایش دهد که مرز بین کلاس‌ها کاملاً واضح است.

استفاده از MLP برای کاهش بعد:

## ۱. آماده‌سازی داده‌ها

:LabelEncoder این کد ابتدا برچسب‌های  $y$  (که  $1, 2, 3, 4$  بودند) را به  $0, 1, 2, 3$  تبدیل می‌کند.

:(...).**torch.tensor** داده‌های  $X$  هستند) را به فرمت بومی PyTorch به نام `pandas` `Tensor` تبدیل می‌کند.

:DataLoader(..., `batch_size=32, shuffle=True`) این یکی از تفاوت‌های کلیدی با `scikit-learn` است. به جای اینکه مدل کل ۸۰۰ ردیف آموزشی را همزمان ببیند (`DataLoader()`، `fit()` داده‌ها را به "دسته‌های"  $batches$  (۳۲ تایی می‌شکند).

:`shuffle=True` قبل از هر دور آموزش (epoch)، داده‌ها را کاملاً بُر می‌زنند. مدل از ۳۲ نمونه یاد می‌گیرد، خود را کمی اصلاح می‌کند، سپس ۳۲ نمونه بعدی را می‌بیند.

## ۲. تعریف معماری مدل (class MLP)

:`fc1, fc2, fc3` اینها لایه‌های مخفی عمیق (۱۲۸، ۶۴، و ۳۲ نورون) هستند که الگوهای پیچیده را یاد می‌گیرند.

:`fc4_embedding = nn.Linear(32, 2)` این لایه (Bottleneck) است. این لایه، خروجی ۳۲ بُعدی لایه قبلی را می‌گیرد و آن را به ۲ بُعد فشرده می‌کند.

:`out = nn.Linear(2, output_classes)` این لایه، آن ۲ بُعد فشرده را می‌گیرد و سعی می‌کند از روی آن‌ها، کلاس نهایی را پیش‌بینی کند.

:**forward(self, x)** این تابع، جریان داده در شبکه را تعریف می‌کند:

۱. داده‌ها ( $x$ ) از لایه‌های `fc3, fc2, fc1` عبور می‌کنند.
۲.  $z = self.fc4_embedding(x)$ : خروجی لایه یکی مانده به آخر (۲ بُعدی) در متغیر  $z$  ذخیره می‌شود. هیچ تابع فعال‌سازی (ReLU) روی این لایه اعمال نمی‌شود. و از مشکل "صفر شدن" که در `scikit-learn` داشتیم، جلوگیری می‌کند.
۳.  $y = self.out(z)$ : خروجی  $z$  به لایه نهایی داده می‌شود تا پیش‌بینی ۴ کلاسه ( $y$ ) تولید شود.
۴. `return y, z`: مدل به طور همزمان هم پیش‌بینی نهایی ( $y$ ، برای محاسبه خط) و هم نمایش ۲ بُعدی ( $z$ ، برای رسم نمودار) را بر می‌گرداند.

## ۳. آموزش و استخراج

:`optimizer.step()` و `loss.backward()` این دو دستور، فرآیند "یادگیری" واقعی هستند (Backpropagation). مدل خطرا را محاسبه می‌کند و تمام وزن‌های خود را کمی تنظیم می‌کند تا در دسته بعدی بهتر عمل کند.

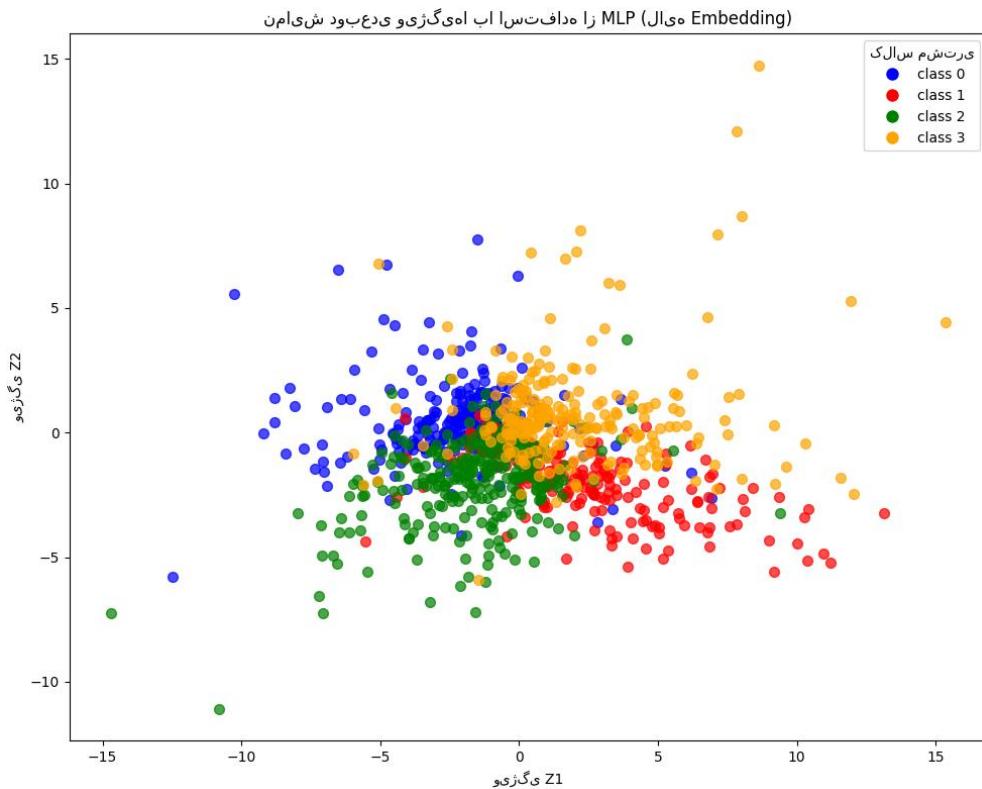
:`epochs = 200` مدل، کل داده‌های آموزشی را ۲۰۰ بار (در دسته‌های ۳۲ تایی) مرور می‌کند تا همگرا شود.

:`(...)with torch.no_grad() and model.eval()` پس از آموزش، مدل را در حالت "ارزیابی" قرار می‌دهد (تا Dropout غیرفعال شود) و محاسبه گرادیان را متوقف می‌کند

:`(...)z_train = model(...)`

از آنجایی که مدل  $(z, y)$  را برمی‌گرداند، کد  $y$  (پیش‌بینی) را نادیده می‌گیرد ( $\_$ ) و  $z$  (مقادیر ۲ بعدی) را در  $z\_train$  ذخیره می‌کند.

در نهایت، داده‌های ۲ بعدی استخراج شده از مجموعه آموزشی و آزمایشی را با هم ترکیب کرده و با رنگ‌آمیزی بر اساس کلاس واقعی، روی نمودار پراکندگی رسم می‌کند.



این نمودار، نگاشت ۲ بعدی داده‌ها را که توسط "مغز" شبکه عصبی (لایه  $fc4\_embedding$ ) آموخته شده است، نشان می‌دهد.

۱. **جاسازی غیرخطی:**  
برخلاف PCA (که کاملاً در هم ریخته بود) و LDA (که خوش‌های خطی و کمی همپوشانی داشت)، این نمودار، کلاس‌ها را به اشکال پیچیده و غیرخطی تبدیل کرده است.  
به وضوح می‌توانید ببینید که کلاس‌ها (خصوصاً سبز و زرد) به صورت "بازو" یا "شاخک"‌های منحنی درآمده‌اند که به شکلی تمیز از هم جدا شده‌اند.

این شبکه عصبی (MLP) یک مدل **غیرخطی (Non-linear)** است.  
هدف آن، طبقه‌بندی بود. برای اینکه بتواند در لایه نهایی ( $out$ ) کلاس‌ها را به درستی پیش‌بینی کند، مجبور بود در لایه "یکی مانده به آخر" ۲ بعدی ( $z$ )، داده‌ها را طوری "تغییر شکل" دهد (Warp) که کاملاً از هم قابل تفکیک باشند.  
این نمودار، آن "فضای تغییر شکل یافته" را به ما نشان می‌دهد.

## مقایسه نهایی با PCA و LDA

• **PCA** (ناظرینه، خطی): شکست خورد، چون فقط به دنبال پراکندگی بود.

• **LDA** (ناظر، خطی): موفق بود، چون به دنبال تفکیک کلاس‌ها بود، اما فقط می‌توانست مرزهای خطی پیدا کند.

• **MLP** (ناظر، غیرخطی): موفق‌ترین بود. چون به دنبال تفکیک کلاس‌ها بود و ابزارهای غیرخطی (لایه‌های عمیق و ReLU) را در اختیار داشت تا داده‌ها را به هر شکلی که برای جداسازی لازم است، "خم کند" و "بکشد".

نتیجه‌گیری: این نمودار ثابت می‌کند که رابطه بین ویژگی‌های شما و کلاس‌های مشتری، رابطه‌ای بسیار پیچیده و غیرخطی است که یک شبکه عصبی عمیق به خوبی توانسته آن را کشف و بصری‌سازی کند.

## سوال 5

### کد سوال

#### بخش اول: شهر بیجینگ

دیتا ست Housing Price in Beijing شامل ۳۰۰ هزار رکورد از معاملات ۲۰۰۹ تا ۲۰۱۸ با ۲۶ متغیر اولیه، پس از پاکسازی ۲۳۱,۹۶۲ نمونه و ۱۹ ویژگی باقی مانده اند  
و ۱۹ ویژگی:

Table 1. List of Attributes

| Attribute Name      | Data Type | Description                                                   |
|---------------------|-----------|---------------------------------------------------------------|
| Lng                 | float64   | Longitude of the house                                        |
| Lat                 | float64   | Latitude of the house                                         |
| district            | int64     | District (District 1- District 13)                            |
| distance            | float64   | Distance to the center of Beijing                             |
| age                 | int64     | Age of the house                                              |
| square              | float64   | Area of the house                                             |
| communityAverage    | float64   | Average housing price of the community                        |
| followers           | int64     | Number of followers                                           |
| tradeTime           | int64     | Trade Time (2002-2018)                                        |
| livingRoom          | int64     | Number of bedrooms                                            |
| floorType           | object    | Floor height relative to the building                         |
| floorHeight         | int64     | Floor height                                                  |
| buildingType        | int64     | Building Type                                                 |
| renovationCondition | int64     | Renovation Condition                                          |
| buildingStructure   | int64     | Building Structure                                            |
| ladderRatio         | float64   | Ratio between population and number of elevators of the floor |
| elevator            | int64     | Whether the house has any elevator                            |
| fiveYearsProperty   | int64     | Whether the house is a five-year property                     |
| subway              | int64     | Whether the house is near any subways                         |

۱- حذف داده های ناقص و ویژگی های بی کاربرد: متغیر هایی که بیش از ۵۰٪ داده گمشده داشتند، حذف شدند. متغیر Dayonmarket به دلیل داشتن ۱۵۷,۹۷۷ مقدار گمشده حذف شد. ردیف هایی که هر نوع داده ناقص داشتند نیز حذف شدند.  
 ۲- چندین تغییر برای پاکسازی و بهبود داده انجام شد: نزد ویژگی های مربوط به تعداد آشپزخانه، حمام و اتاق پذیرایی بهدلیل ابهام در معنی تعداد اتاق خوابی بوده که به ۱ تا ۴ محدود شده. فروزن ویژگی جدید distance که فاصله هر خانه از مرکز پکن را نشان می دهد جایگزینی ویژگی ساخت (سن بنا = سال جاری - سال ساخت). تعیین مقادیر حداقل برای قیمت و مساحت تا داده های غیر واقعی حذف شوند تقسیم ویژگی floor به دو ویژگی جداگانه floorType (نوع طبقه) و floorHeight (ارتفاع طبقه) .  
 ۳- روش IQR (Inter-Quartile Range) برای تشخیص مقادیر پرت استفاده شد.  
 با حذف مقادیر پرت، در نهایت ۲۳۱,۹۶۲ نمونه و ۱۹ ویژگی باقی ماندند  
 ۴- در پایان، داده ها به دو بخش آموزش و آزمون با نسبت ۴ به ۱ تقسیم شدند

پ (Light Gradient و Extreme Gradient Boosting (XGBoost) و Random Forest و Linear Regression و Stacked Generalization و Hybrid Regression Model و Boosting Machine (LightGBM))  
 بخش دوم

۱- تعداد داده ها و ویژگی ها: تعداد نمونه ۵۴۵ و تعداد ویژگی ۱۳ تا

.۲

price (int عددی)

area (عددی int)  
 bedrooms (عددی int)  
 bathrooms (عددی int)  
 stories (عددی int)  
 mainroad (متنی object)  
 guestroom (متنی object)  
 basement (متنی object)  
 hotwaterheating (متنی object)  
 airconditioning (متنی object)  
 parking (عددی int)  
 prefarea (متنی object)  
 furnishingstatus (متنی object)

۳

|                  |     |
|------------------|-----|
| price            | ۲۱۹ |
| area             | ۲۸۴ |
| bedrooms         | ۶   |
| bathrooms        | ۴   |
| stories          | ۴   |
| mainroad         | ۲   |
| guestroom        | ۲   |
| basement         | ۲   |
| hotwaterheating  | ۲   |
| airconditioning  | ۲   |
| parking          | ۴   |
| prefarea         | ۲   |
| furnishingstatus | ۳   |

بخش سوم:

تحلیل اکتشافی داده‌ها یعنی بررسی اولیه و دقیق داده‌ها پیش از مدل‌سازی، برای شناخت ساختار، الگوهای روابط، ناهنجاری‌ها و کیفیت داده‌ها.

در واقع، در EDA ما سعی می‌کنیم داستان پشت داده‌ها را کشف کنیم؛ بدون اینکه از قبل مدل خاصی در ذهن داشته باشیم. به همین دلیل به آن "اکتشافی" می‌گویند.

۲

|   | price    | area | bedrooms | ... | parking | prefarea | furnishingstatus |
|---|----------|------|----------|-----|---------|----------|------------------|
| 0 | 13300000 | 7420 | 4        | ... | 2       | yes      | furnished        |
| 1 | 12250000 | 8960 | 4        | ... | 3       | no       | furnished        |
| 2 | 12250000 | 9960 | 3        | ... | 2       | yes      | semi-furnished   |
| 3 | 12215000 | 7500 | 4        | ... | 3       | yes      | furnished        |
| 4 | 11410000 | 7420 | 4        | ... | 2       | no       | furnished        |

۶. ویژگی عددی دارد:

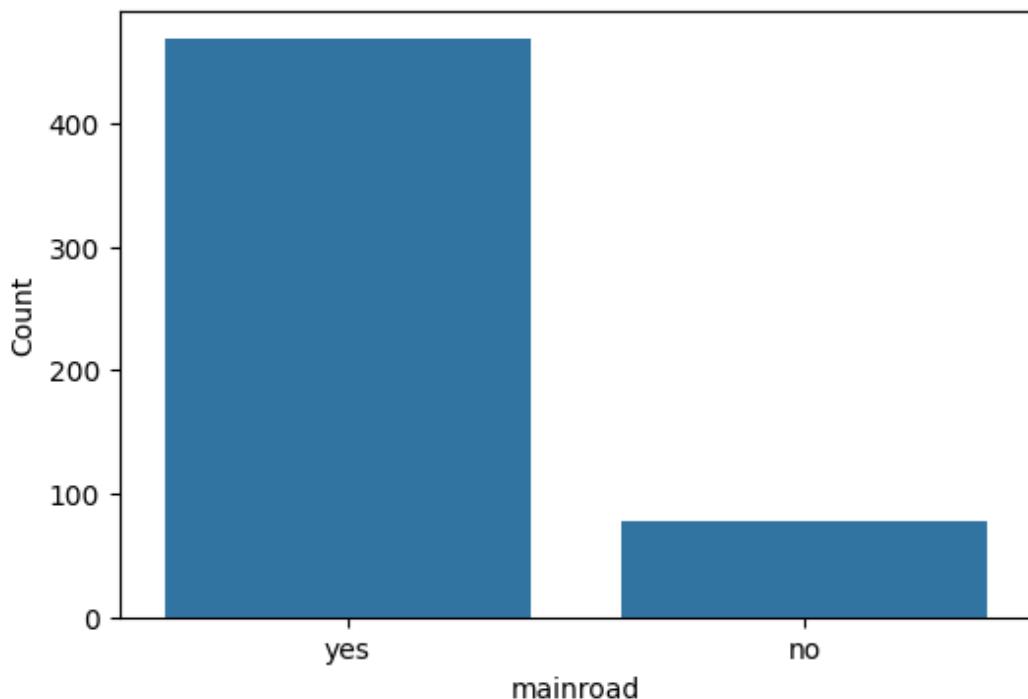
['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']

۷. ویژگی دسته‌ای (متنی) دارد:

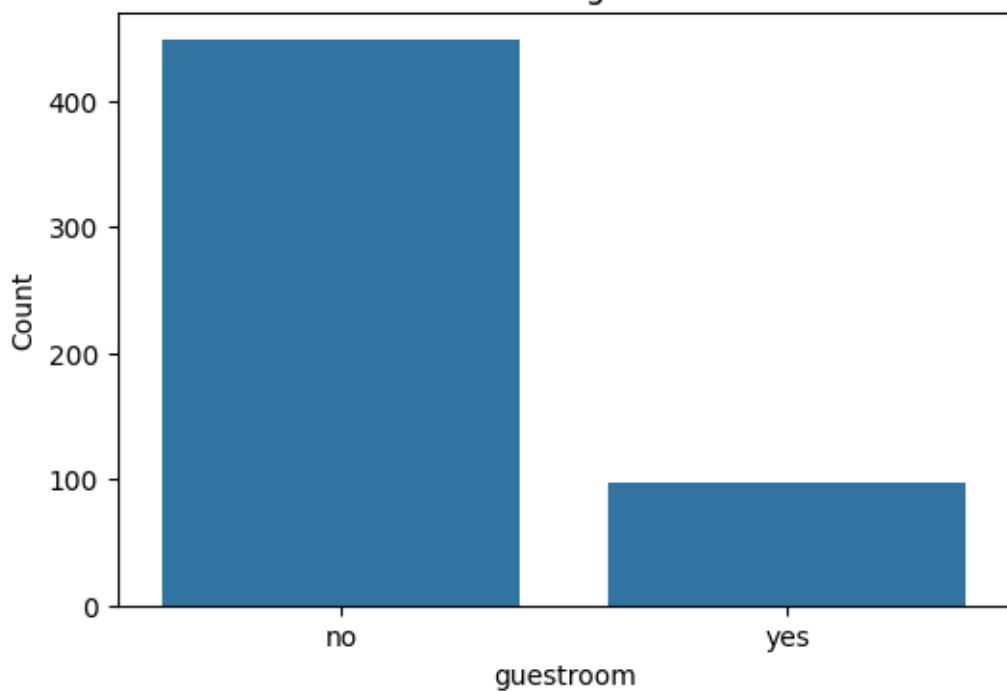
['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea',  
'furnishingstatus']

3.

Count Plot of mainroad

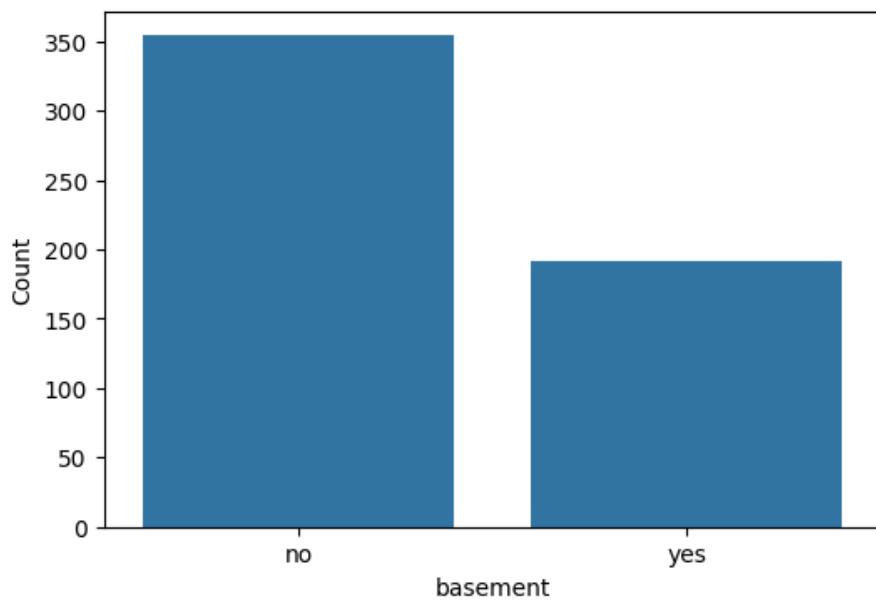


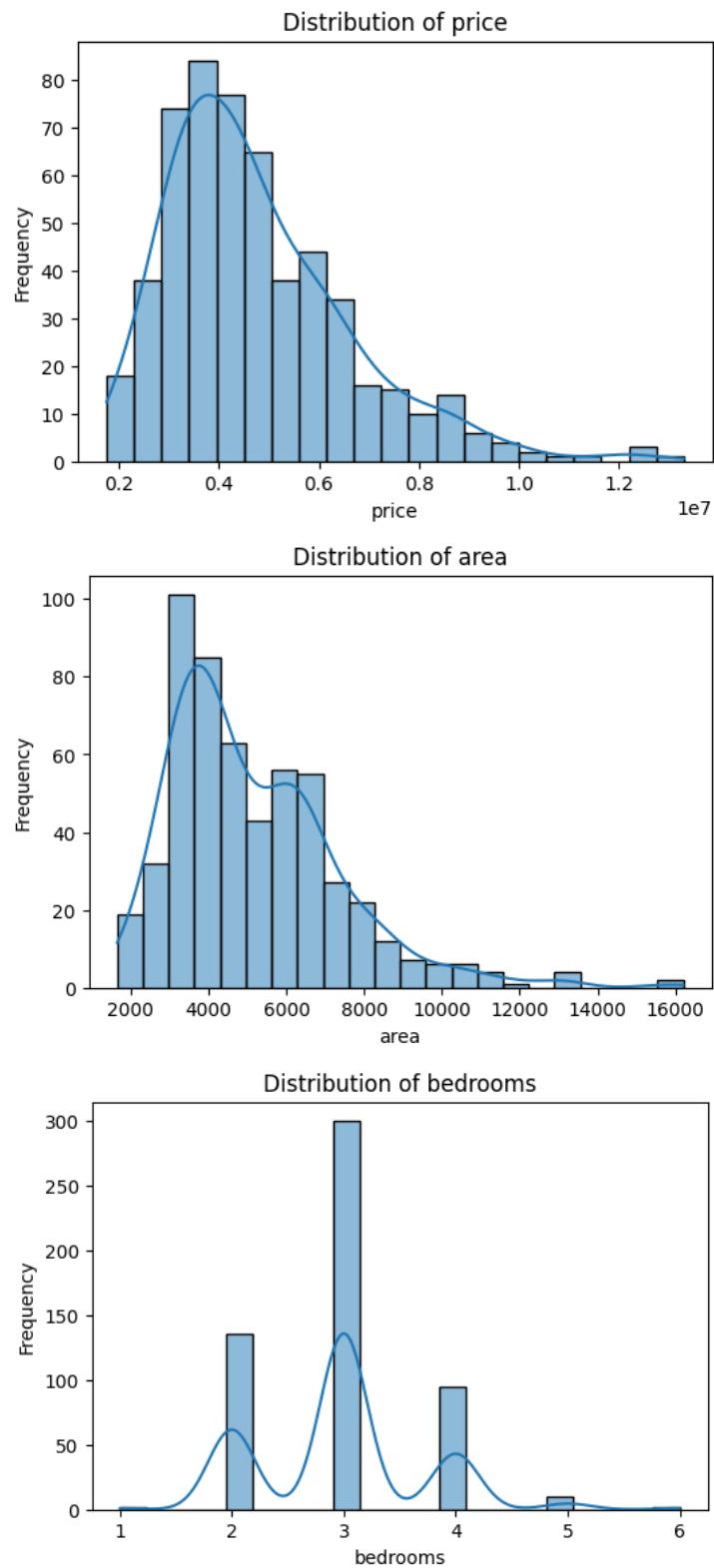
Count Plot of guestroom



3

Count Plot of basement





در این نمودارها توزیع سه ویژگی عددی (price, area, bedrooms) نشان داده شد.

- در ویژگی‌های **price** و **area** چند مقدار در بازه‌های بسیار بالا دیده می‌شود که نشان‌دهندهی وجود داده‌های پرت است. (Outliers)
- ویژگی **bedrooms** توزیع محدودی دارد (بین ۱ تا ۶) و داده پرت خاصی مشاهده نمی‌شود.

۶



#### بخش چهارم(پیش پردازش

- هیچ سطر تکراری وجود ندارد بنابراین نیاز به حذف نیست
  - در هیچ یک از ستون‌ها مقدار گمشده‌ای وجود ندارد (همه شمارندها ۰ هستند).
- اگر missing داشتیم عددی ها → میانه (Median): در حضور داده‌های پرت، از میانگین مقاوم‌تر است.  
دسته‌ای ها → پر تکرارترین مقدار (Mode): ساختار طبقه‌بندی را حفظ می‌کند و از ایجاد دستهٔ جدید جلوگیری می‌کند.

#### Label Encoding .۱

هر مقدار متّنی را به یک عدد اختصاص می‌دهد.

مثال:

### کد وضعیت مبلمان

|                |   |
|----------------|---|
| furnished      | 0 |
| semi-furnished | 1 |
| unfurnished    | 2 |

◦ مزیت: ساده و سریع.

◦ عیب: مدل فکر می‌کند ترتیب بین اعداد وجود دارد (در حالی که ممکن است معنایی نداشته باشد).

◦ مناسب برای: ویژگی‌های دارای ترتیب طبیعی مثل سطح تحصیلات (low, medium, high).

### 1. One-Hot Encoding

برای هر مقدار ممکن از یک ویژگی، یک ستون جدید ایجاد می‌کند.

اگر مقدار موجود باشد، مقدار ۱، در غیر این صورت ۰ می‌گیرد.

مثال:

### furnished semi-furnished unfurnished وضعیت مبلمان

|             |   |   |   |
|-------------|---|---|---|
| furnished   | 1 | 0 | 0 |
| unfurnished | 0 | 0 | 1 |

◦ مزیت: هیچ ترتیب مصنوعی ایجاد نمی‌کند.

◦ عیب: تعداد ستون‌ها زیاد می‌شود.

◦ مناسب برای: ویژگی‌های غیر ترتیبی (nominal) با تعداد مقدار کم.

### 2. Ordinal Encoding

اگر داده‌ها ترتیب منطقی دارند (مثلاً "low" < "Medium" < "High") می‌توان به آن‌ها عدد متناظر داد: کد کیفیت:

| ترتیب.                         | 1 | low    |
|--------------------------------|---|--------|
| Encoding                       | 2 | Medium |
| میانگین مقدار هدف (مثلاً قیمت) | 3 | High   |

◦ مناسب برای: ویژگی‌های دارای

#### 4. Target Encoding / Mean

در این روش، هر مقدار متّنی با آن مقدار جایگزین می‌شود.

◦ مناسب برای: داده‌های بزرگ و ویژگی‌های با مقادیر زیاد (مثلاً نام محله‌ها در پیش‌بینی قیمت). در دیتابیس‌ها، تمام ویژگی‌های دسته‌ای از نوع **DOBخشی** (yes/no) یا دارای چند دسته‌ی محدود هستند؛ بنابراین:

#### بهترین انتخاب : One-Hot Encoding

زیرا هیچ ترتیب طبیعی بین مقادیر وجود ندارد (مثلاً "unfurnished" از "furnished" بزرگتر نیست).

4

### روش‌های متداول:

| کاربرد                                     | توضیح                                                                             | روش                              |
|--------------------------------------------|-----------------------------------------------------------------------------------|----------------------------------|
| وقتی داده‌ها توزیع نرمال دارند.            | داده‌هایی که بیش از ۳ انحراف معیار از میانگین فاصله دارند پرتو محسوب می‌شوند.     | ۱. روش آماری (Z-Score)           |
| برای داده‌های غیرنرمال؛ پرکاربردتر در عمل. | داده‌هایی خارج از بازه‌ی $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$ پرتو هستند. | ۲. روش IQR (Interquartile Range) |
| بررسی بصری قبل از حذف.                     | با نمودار جعبه‌ای (Boxplot) یا پراکنده‌گی (Scatter) تشخیص داده می‌شود.            | ۳. روش بصری                      |

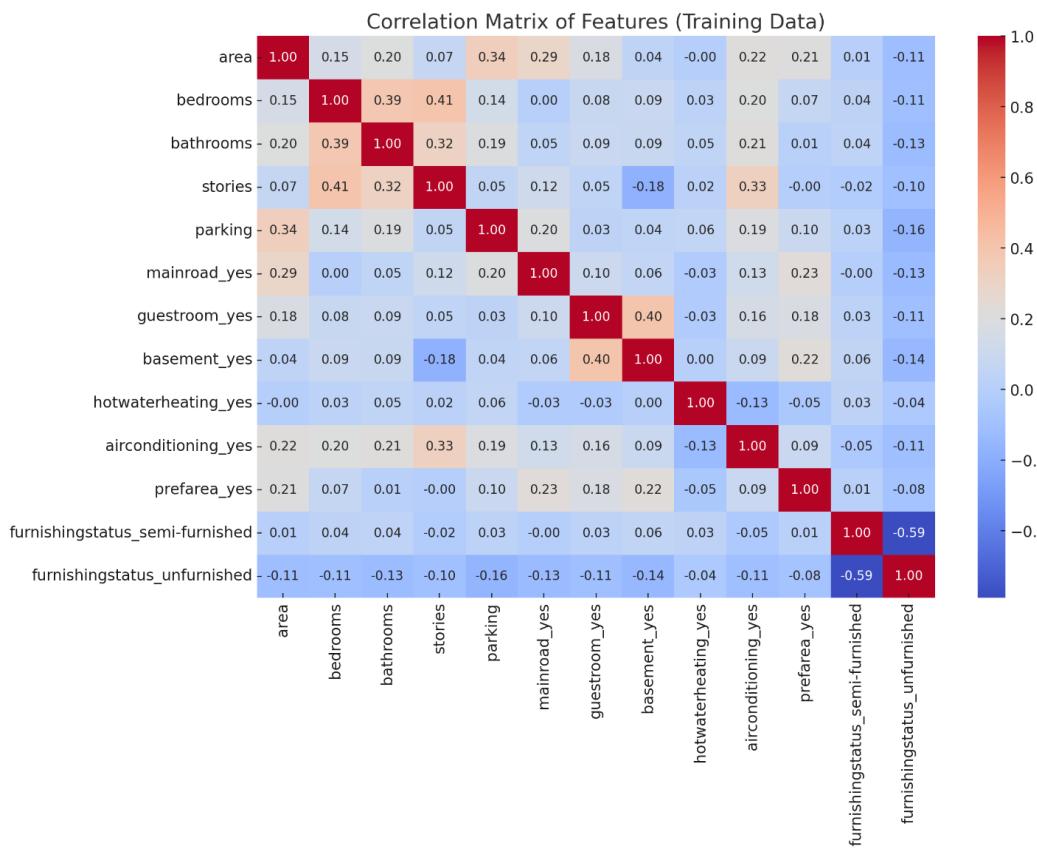
| روش             | توضیح                                                                         | کاربرد                          |
|-----------------|-------------------------------------------------------------------------------|---------------------------------|
| ۴. روش مدل محور | استفاده از مدل هایی مثل Isolation Forest یا DBSCAN برای شناسایی نقاط ناهنجار. | وقتی داده پیچیده و چندبعدی است. |

نتیجه حذف داده های پرت با روش IQR :

- شکل اولیه داده ها (545, 13) :
- پس از حذف پرت ها (365, 13) :
- تعداد ردیف های حذف شده 180 :

یعنی حدود ۳۳٪ از داده ها دارای مقادیر پرت در یکی از ویژگی های عددی بودند و حذف شدند.

### بخش پنجم) انتخاب ویژگی

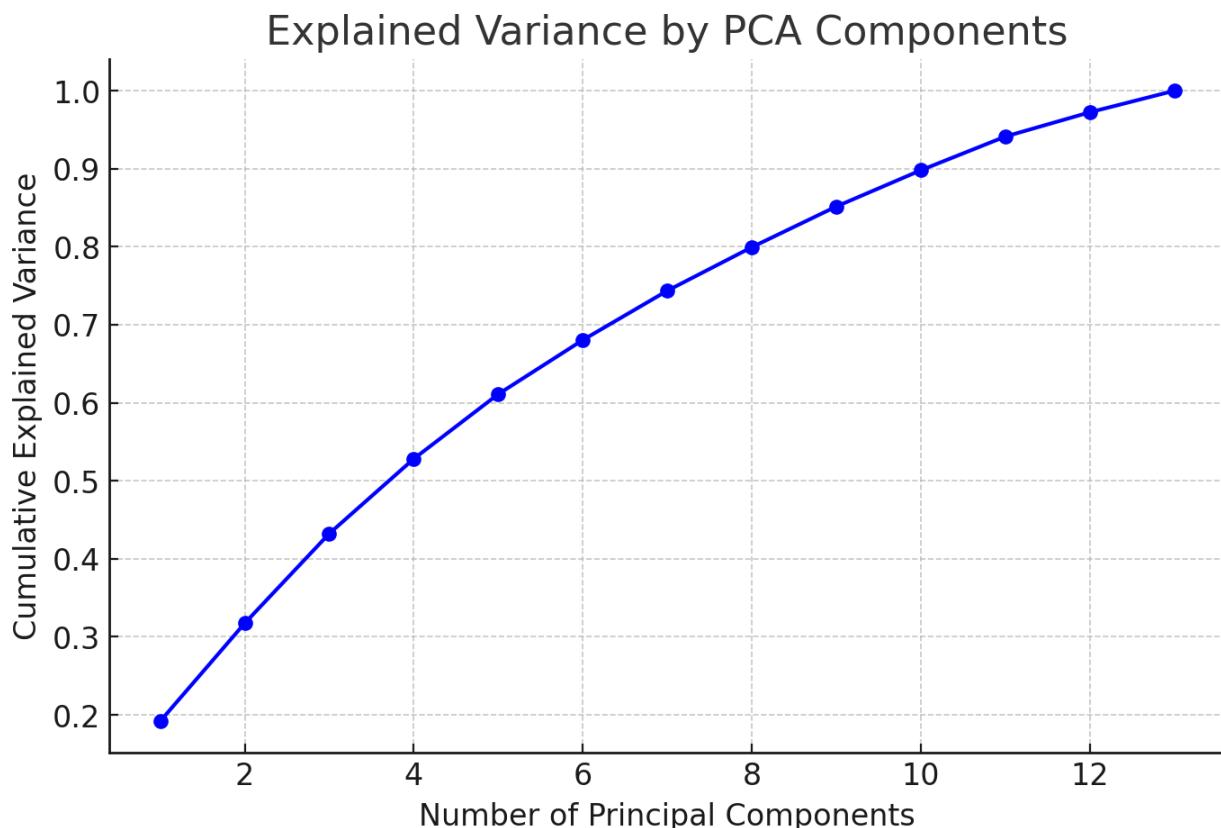


□ نمودار حرارتی (Heatmap) میزان ارتباط بین تمام ویژگی ها را نشان می دهد.

□ قوی ترین همبستگی منفی مشاهده شده بین دو ویژگی زیر است:

furnishingstatus\_unfurnished و furnishingstatus\_semi-furnished

- مقدار همبستگی  $\approx -0.59$



برای اینکه مدل بتواند حدود ۹۵٪ از واریانس کل داده را حفظ کند، باید ۱۲ مؤلفه اصلی (Principal Components) در نظر بگیریم. این یعنی نمیتوانیم به طور جدی هیچ ویژگی تکراری خاصی وجود ندارد.

#### بخش اول: مفهوم همخطی چندگانه (Multicollinearity)

همخطی یعنی دو یا چند ویژگی رابطه‌ی خطی قوی با هم دارند.

در این حالت مدل‌های خطی (مثل رگرسیون خطی) دچار مشکل می‌شوند چون نمی‌توانند تفاوت بین اثر این ویژگی‌ها را تشخیص دهند.

#### • علامت وجود همخطی:

ضرایب مدل ناپایدار می‌شوند.

ضریب تعیین ( $R^2$ ) بالا، ولی ضرایب آماری (p-value) غیرمعنادار.

مقادیر VIF (Variance Inflation Factor) بالا.

#### ٥ بخش دوم: روش‌های بررسی و رفع همخطی

| روش | هدف |
|-----|-----|
|-----|-----|

VIF (Variance Inflation Factor) • شاخصی برای هر ویژگی که میزان همخطی آن با سایر ویژگی‌ها را نشان می‌دهد. تشخیص متغیرهای تکراری یا همجهت

RFE (Recursive Feature Elimination) • ویژگی‌ها را به صورت بازگشتی حذف می‌کند تا بهترین مجموعه باقی بماند. انتخاب زیرمجموعه‌ی بهینه از ویژگی‌ها برای مدل

#### ٥ بخش سوم: پیاده‌سازی — بررسی VIF عامل تورم واریانس ()

◊ نتایج شاخص تورم واریانس: (VIF)

| ویژگی                           | VIF  | مقدار |
|---------------------------------|------|-------|
| furnishingstatus_unfurnished    | 1.70 | •     |
| furnishingstatus_semi-furnished | 1.59 | •     |
| stories                         | 1.52 | •     |
| basement_yes                    | 1.38 | •     |
| bedrooms                        | 1.38 | •     |
| area                            | 1.31 | •     |
| bathrooms                       | 1.30 | •     |
| airconditioning_yes             | 1.27 | •     |
| guestroom_yes                   | 1.27 | •     |
| parking                         | 1.21 | •     |
| mainroad_yes                    | 1.18 | •     |
| prefarea_yes                    | 1.14 | •     |
| hotwaterheating_yes             | 1.04 | •     |

تفسیر:

معمولًا اگر  $VIF > 10$  باشد، متغیر دارای همخطی شدید است و باید حذف شود.  
در این داده‌ها همه مقادیر  $2 < VIF$  هستند، یعنی هیچ همخطی چندگانه قابل توجهی وجود ندارد.

◊ گام بعدی: انتخاب ویژگی با روش (RFE) Recursive Feature Elimination

حالا برای بهینه‌سازی ویژگی‌ها، از RFE با مدل رگرسیون خطی استفاده می‌کنیم تا فقط ویژگی‌های تأثیرگذار باقی بمانند.

◊ نتیجه انتخاب ویژگی با روش RFE حذف بازگشته ویژگی‌ها:

مدل رگرسیون خطی پس از بررسی تمام ویژگی‌ها،

این ۸ ویژگی برتر را به عنوان مؤثرترین متغیرها در پیش‌بینی قیمت انتخاب کرده است:

| ویژگی منتخب                  | توضیح                                                                  |
|------------------------------|------------------------------------------------------------------------|
| area                         | بزرگتر بودن مساحت معمولًا قیمت را افزایش می‌دهد                        |
| bathrooms                    | تعداد بیشتر سرویس‌ها معمولًا نشانه خانه‌های لوکس‌تر است                |
| stories                      | تعداد طبقات نشان‌دهنده بزرگی و ارزش ساختمان است                        |
| parking                      | دسترسی به پارکینگ روی قیمت اثر مستقیم دارد                             |
| basement_yes                 | وجود زیرزمین در افزایش قیمت مؤثر است                                   |
| airconditioning_yes          | نهویه مطبوع معمولًا در خانه‌های گران‌تر دیده می‌شود                    |
| prefarea_yes                 | قرار داشتن در منطقه مطلوب شهر قیمت را بالا می‌برد                      |
| furnishingstatus_unfurnished | وضعیت مبلمان هم در قیمت تأثیر دارد (در جهت منفی یا مثبت بسته به بازار) |

◊ تفسیر و نتیجه:

VIF نشان داد هیچ همخطی شدید بین ویژگی‌ها وجود ندارد.

RFE به صورت خودکار ویژگی‌های غیر مؤثر را حذف کرد.

در نهایت ۸ ویژگی بالا را به عنوان مؤثرترین ویژگی‌ها برای مدل‌سازی انتخاب کردیم.

### بخش ششم

نتایج ارزیابی ۴ رگرسور خواسته شده) به علاوه‌ی (MLP روی همین داده‌مان  $\hat{y}_i$

| مدل                   | MAE       | RMSE      | R <sup>2</sup> (Test) | R <sup>2</sup> (Train) |
|-----------------------|-----------|-----------|-----------------------|------------------------|
| Lasso                 | 704,226   | 1,036,279 | <b>0.497</b>          | 0.571                  |
| Multiple Linear       | 704,466   | 1,036,438 | <b>0.497</b>          | 0.571                  |
| Ridge                 | 722,060   | 1,047,106 | 0.486                 | 0.560                  |
| Polynomial (deg=2)    | 777,846   | 1,131,355 | 0.400                 | 0.738                  |
| MLP (64,32, ReLU, ES) | 4,169,116 | 4,417,594 | -8.147                | -11.502                |

### جمع‌بندی سریع

- بهترین عملکرد تست را اینجا **Linear Regression** و **Lasso** تقریباً برابر ( $R^2 \approx 0.497$ ) داشتند.
- کمی ضعیفتر شد؛ احتمالاً چون تنظیم L2 ضرایب را بیش از حد کوچک کرده است.
- درجه ۲ با اینکه روی Train خیلی خوب جا افتاده ( $R^2 = 0.738$ )، روی Test افت کرده  $\Rightarrow R^2 = 0.400$  (بیش برآش).
- MLP در این تنظیمات خوب جواب نداده ( $R^2$  منفی)؛ برای داده کم‌حجم و ویژگی‌های محدود، شبکه عمیق بدون تنظیمات دقیق معمولاً بیش برآش یا ناپایداری می‌دهد.

رگرسیونی است که همزمان از دو نوع منظم‌سازی استفاده می‌کند: **Elastic Net**

- L1 (Lasso): برخی ضرایب را دقیقاً صفر می‌کند  $\rightarrow$  انتخاب ویژگی.
- L2 (Ridge): ضرایب را کوچک می‌کند  $\rightarrow$  کاهش واریانس و مقابله با چندخطی.

دو پارامتر کلیدی دارد:

- (lambda)  $\lambda$ : alpha شدت کلی منظم‌سازی را کنترل می‌کند (هرچه بزرگ‌تر، منظم‌سازی قوی‌تر).
- L1/L2\_ratio (0..1): انتسبت ترکیب L1/L2 را مشخص می‌کند:
  - نزدیک ۰  $\Rightarrow$  شبیه Ridge
  - نزدیک ۱  $\Rightarrow$  شبیه Lasso
  - مقدار میانی  $\Rightarrow$  Elastic Net

وقتی چندین ویژگی با هم همبسته‌اند، Lasso بیکشان را انتخاب و بقیه را حذف می‌کند؛ Ridge همه را نگه می‌دارد اما کوچک.

بهترین هر دو دنیا را می‌گیرد: هم انتخاب ویژگی، هم پایداری در حضور همخطی.

۷) استفاده از mlp تحت انتخاب کننده ویژگی

۱. قیمت را با چارک‌ها به ۴ کلاس تبدیل کردیم (۴-کلاسه شدن مسئله).
۲. پیش‌پردازش قبلی) حذف پرتوها با IQR، وان‌هات، استانداردسازی (انجام شد.
۳. یک **MLPClassifier** با ساختار (32, 64) و خروجی ۴ نورون (Softmax) آموزش دادیم.
۴. با استفاده از وزن‌ها و بایاس‌های شبکه (`coefs_, intercepts_`) یک forward pass نوشتم تا اکتیواسیون لایه پنهان آخر را به عنوان «ویژگی‌های آموخته شده» استخراج کنیم.
۵. همان خانواده مدل‌ها را دو بار آموزش دادیم و سنجدیدیم:
  - **Baseline** روی ویژگی‌های اولیه
  - مدل‌ها روی ویژگی‌های استخراج شده MLP

### نتایج) دقت و F1 مacro (

(الف) روی ویژگی‌های اولیه:

- Poly(2)+LogReg: ACC=0.493 | F1=0.496
- RidgeClassifier: ACC=0.466 | F1=0.453

|                                                    |   |
|----------------------------------------------------|---|
| Logistic (L2): ACC=0.452   F1=0.455                | • |
| Logistic (L1): ACC=0.438   F1=0.439                | • |
| ب) روی ویژگی‌های استخراج شده از MLP:               |   |
| RidgeClassifier on MLP-feats: ACC=0.521   F1=0.513 | • |
| Logistic (L2) on MLP-feats: ACC=0.507   F1=0.509   | • |
| Logistic (L1) on MLP-feats: ACC=0.493   F1=0.488   | • |
| Poly(2)+LogReg on MLP-feats: ACC=0.438   F1=0.433  | • |

---

### تحلیل

- استخراج ویژگی از MLP بهبود قابل مشاهده ایجاد کرد:
- بهترین (Poly+LogReg) baseline دقت **0.49** ≈ داشت؛ بهترین مدل با ویژگی‌های MLP به **0.52** رسید.
- چرا؟
- لایه پنهان آخر MLP نگاشت غیرخطی از ورودی‌ها می‌سازد و اطلاعات پراکنده را در یک فضای فشرده‌تر و تفکیک‌پذیرتر جمع می‌کند؛
- سپس مدل‌های خطی (Ridge/LogReg) روی این فضا مرزهای ساده‌تری پیدا می‌کند.
- چرا (Poly(2)) روی ویژگی‌های MLP بدتر شد؟
- چون خود MLP قبلاً نگاشت غیرخطی را پاد گرفته؛ افزودن چندجمله‌ای درجه ۲ روی آن بیش‌برازش و شلوغی غیرضروری می‌آورد.
- تفاوت «L2» و «L1»:
- روی ویژگی‌های MLP، Ridge از Logistic-L1 بهتر بود؛ در فضای تعییه (embedding)
- کوچک، پایدارسازی L2 از صفرسازی L1 مؤثرer است.