

گزارش مبانی سیستم های هوشمند

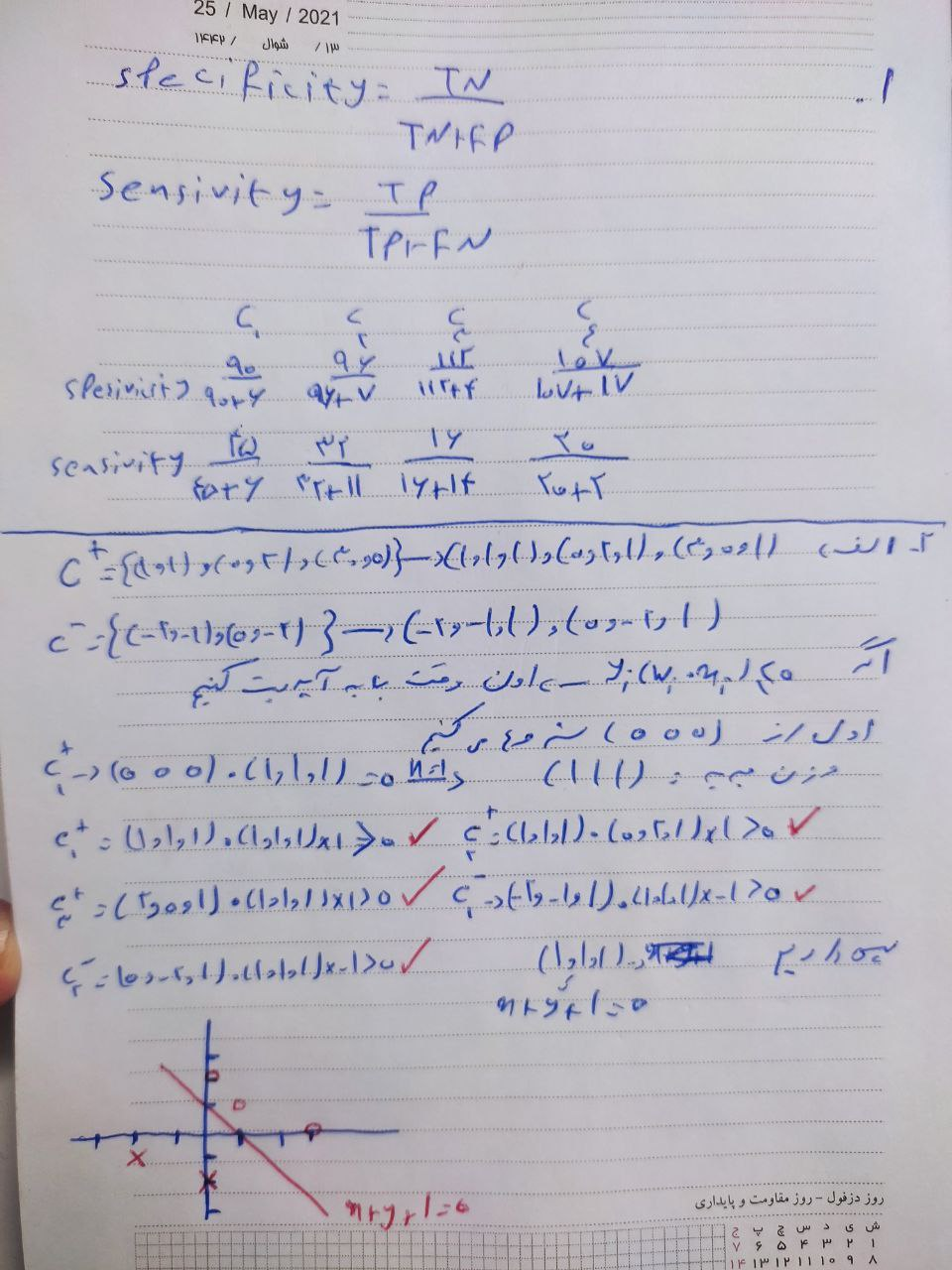
ارشیا کلانتریان 40121993

محمد حسین گل محمدی 40220903

مینی پروژه اول

پاییز ۱۴۰۴

# سوال 1و2



A notebook with writing on it

AI-generated content may be incorrect.

A piece of paper with writing on it

AI-generated content may be incorrect.

## 3-2-

روش فیشر و کمترین مربعات تقریبا یک خط شدند و جداسازی بهتری به ما میدهند در mlp به دلیل اینکه فقط یک بار ان هم با میو 1 انجام شده جدا سازی بهینه نیست

A graph with lines and dots

AI-generated content may be incorrect.

# سوال3

## 1-3-

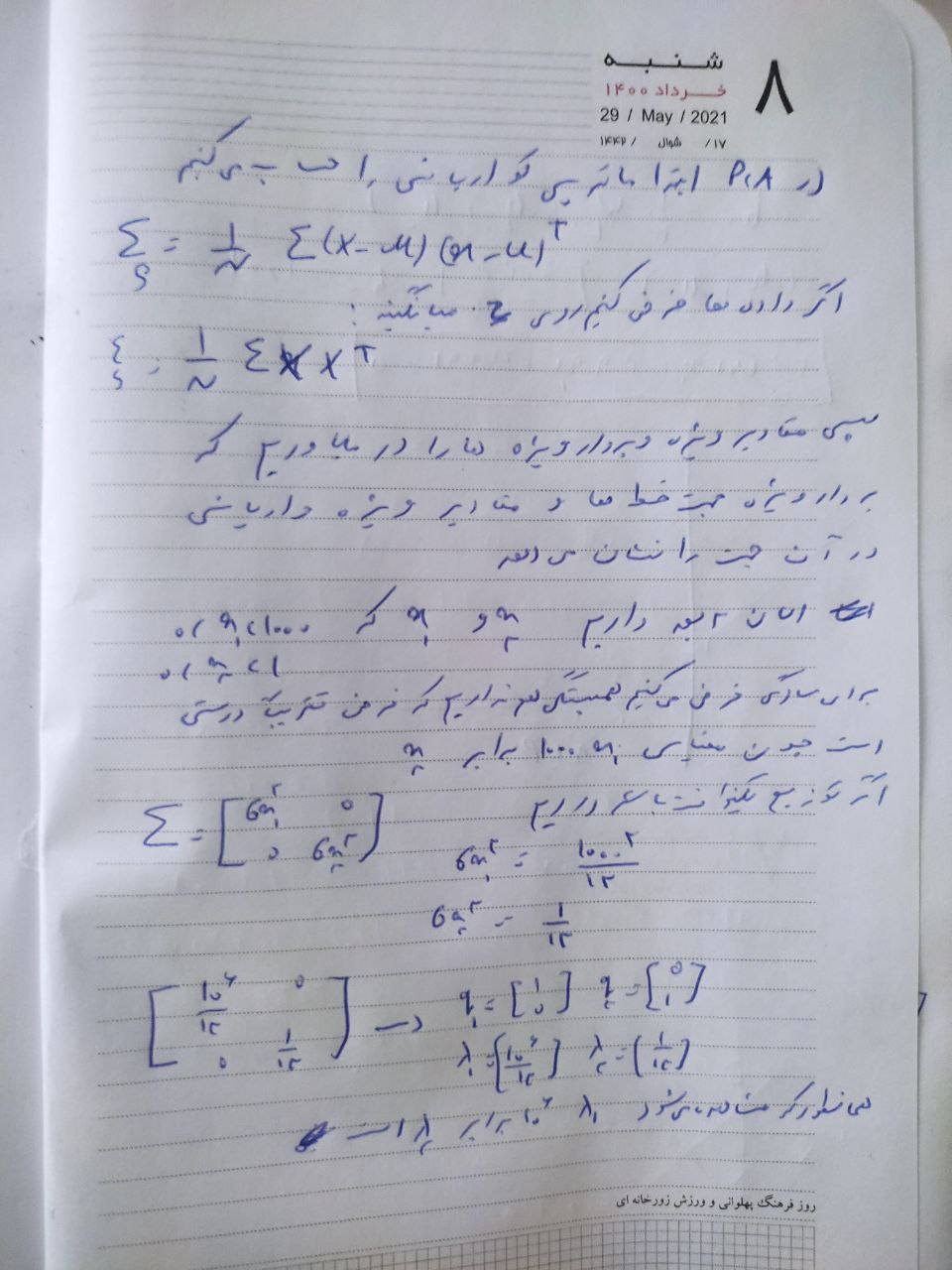
چون pca بر مبنای کوواریانس کار میکند چون واریانس x1 تقریبا 10 به نوان 6 برابر واریانس x2 هست تقریبا پراکندگی به طور کامل در راستای x1 قرار خواهد داشت (تقریبا خط صاف میشه)

## 2-3-

گر ویژگی‌ها در مقیاس‌های بسیار متفاوت باشند، کوواریانسِ ویژگی با دامنه‌ی بزرگ‌تر مقدار عددی خیلی بزرگ‌تری خواهد داشت → مقدار ویژه‌ی بزرگ‌تر → PCA آن را مهم‌تر می‌داند.  
بنابراین PCA جهت مؤلفه اصلی را تقریباً در راستای ویژگی بزرگ‌تر (در اینجا انتخاب می‌کند)، حتی اگر از نظر «اطلاعاتی» هر دو ویژگی مهم باشند.

## 3-3-

.مرکز کردن داده ها: قبل از اجرای PCA باید مقدار میانگین هر ویژگی را از خودش کم کنیم تا داده‌ها دورِ محور صفر قرار بگیرند.2: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



# سوال 4

[کد سوال 4](https://colab.research.google.com/drive/1C9OoH_Zrjj4uAOTx_dhFP7iL61cbQZBe#scrollTo=fMujfs__kpRp)

## 1-4-

### **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:

1. **بررسی ساختاری** (با .info())
2. **بررسی آماری** (با .describe())
3. **بررسی مقادیر گمشده** (با .isnull())

**توضیح کد (بلوک 4.1)**

1. **try...except...else**:
   * **try**: کد سعی می‌کند فایل csv را با pd.read\_csv(output\_file) بخواند و آن را در متغیر df ذخیره کند.
   * **except FileNotFoundError**: این یک مدیریت خطا است. اگر فایل در مرحله قبل به درستی دانلود نشده باشد، برنامه به جای "crash" کردن، یک پیام خطا چاپ می‌کند.
   * **else**: این بلوک فقط در صورتی اجرا می‌شود که try موفقیت‌آمیز بوده باشد. این تضمین می‌کند که کدهای تحلیلی (.info(), .describe()) فقط روی داده‌هایی اجرا می‌شوند که با موفقیت بارگیری شده‌اند.
2. **df.info()**:
   * این تابع یک **خلاصه فنی** از DataFrame به شما می‌دهد.
   * **خروجی کلیدی:**
     + تعداد کل ردیف‌ها (Entries) و ستون‌ها (Data columns).
     + نام، تعداد مقادیر **غیر گمشده (Non-Null)** و **نوع داده (Dtype)** (مثلاً 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 ستون‌ها ۱۰۰۰ است. که یعنی **داده گمشده (NaN) نداریم**.
3. **df.describe()** :
   * این تابع یک **خلاصه آماری** سریع از ستون‌های **عددی** ارائه می‌دهد.
   * **خروجی کلیدی:** mean (میانگین)، std (انحراف معیار)، min (حداقل)، max (حداکثر) و چارک‌ها (25%، 50%، 75%).
   * این خروجی به ما کمک کرد تا **مقیاس** داده‌ها را ببینیم. مثلاً 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 داده‌ها ---

try:

    df = pd.read\_csv(output\_file)

    print("فایل teleCust1000t.csv با موفقیت بارگذاری شد.\n")

except FileNotFoundError:

    print("خطا: فایل دانلود نشد. لطفاً اتصال اینترنت را بررسی کنید.")

else:

    # --- سوال ۱: استفاده از .info() و .describe() ---

    print("--- خروجی تابع .info() ---")

    # .info() ساختار کلی، نوع داده‌ها و تعداد مقادیر غیر گمشده را نشان می‌دهد

    df.info()

    print("\n" + "="\*40 + "\n")

    print("--- خروجی تابع .describe() ---")

    # .describe() خلاصه‌ی آماری (میانگین، انحراف معیار، چارک‌ها و ...)

    # را فقط برای ستون‌های عددی نمایش می‌دهد

    print(df.describe())

    print("\n" + "="\*40 + "\n")

    # --- سوال ۲: بررسی وجود داده‌های گمشده (NaN) ---

    print("--- بررسی تعداد داده‌های گمشده (NaN) در هر ستون ---")

    nan\_counts = df.isnull().sum()

    print(nan\_counts)

    print("\n")

    if nan\_counts.sum() == 0:

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

    else:

        print(f"نتیجه: مجموعاً {nan\_counts.sum()} داده گمشده پیدا شد.")

**خروجی کد:**

فایل teleCust1000t.csv با موفقیت بارگذاری شد.

--- 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 1000.0000 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 1000.000000 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 1000.000000 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

========================================

--- بررسی تعداد داده‌های گمشده (NaN) در هر ستون ---

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

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

**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:* گفتن "میانگین age (سن) ۴۱ سال است" یا "بیشترین income ۱۶۶۸ واحد است" منطقی است.
* **ویژگی‌های طبقه‌ای (Categorical):**
  + این ویژگی‌ها برچسب‌هایی 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.
  + *مثلا* ستون region (منطقه) دارای مقادیر ۱، ۲، ۳ است. محاسبه میانگین منطقه یعنی (mean=2.0220 در خروجی describe) 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: (مدت 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 تاهل) - برچسبی برای ۲ گروه (مقادیر: ۰, ۱).
* ed: (سطح تحصیلات) - برچسبی برای ۵ گروه (مقادیر: ۱, 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 دیتاست است. به زبان ساده، این کد به سه سوال پاسخ می‌دهد:

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

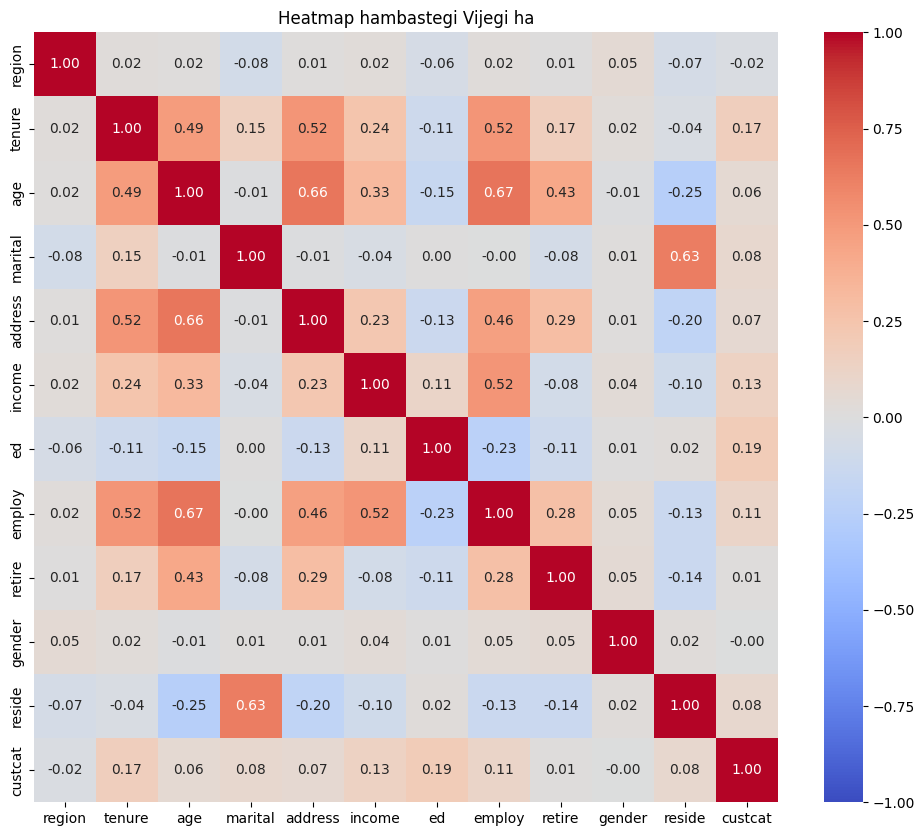
عملکرد خط به خط

1. 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 ۰ به معنای عدم همبستگی *خطی* است.
2. sns.heatmap(corr\_matrix, ...):
   * از کتابخانه seaborn برای رسم یک "نقشه حرارتی" از این ماتریس استفاده می‌کنیم.
   * annot=True: این دستور به seaborn می‌گوید که عدد همبستگی را داخل هر مربع رنگی بنویسد.
   * fmt='.2f': 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.
3. custcat\_corr = ...:
   * . ستون مربوط به متغیر هدف (custcat) را جدا می‌کنیم.
   * .sort\_values(ascending=False): مرتب سازی نتایح یرای دیدن اینکه کدام ویژگی‌ها قوی‌ترین تأثیر را بر روی custcat دارند.

نتیجه کد:

* ویژگی 'ed' ... با 19% ... بیشترین همبستگی را دارد این نشان می‌دهد که مدت زمان اشتراک، قوی‌ترین *پیش‌بینی‌کننده خطی* برای دسته مشتری است.
* بالاترین همبستگی فقط 19% است که بسیار ضعیف محسوب می‌شود پس تقریبا هیچ‌کدام همبستگی خطی *خیلی قوی* ندارند.

**نقشه Heatmap:**



**توضیح کد (بلوک 4.1.4 PairPlot)**

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

**عملکرد کد**

به کمک کتابخانه seaborn یک شبکه از نمودار ها ایجاد کردیم و 4 ویژگی **['tenure', 'age', 'income', 'employ']**: را برای رسم نمودار درنظر گرفتیم و میبینیم که آیا کلاس های مختلف در الگو های متفاوتی ظاهر میشوند یا خیر

**تحلیل نمودار و تفسیر**

**خروجی کد:**

****

تحلیل خروجی:

1. **همپوشانی شدید کلاس‌ها (رنگ‌ها):**
   * به هر کدام از نمودارهای پراکندگی که نگاه کنیم (مثلاً age در برابر income یا tenure در برابر employ)، می‌بینید که هر چهار رنگ (نماینده ۴ کلاس مشتری) به شدت در هم آمیخته‌اند.
   * **هیچ** مرز واضحی وجود ندارد. شما نمی‌توانید یک خط بکشید یا یک دایره رسم کنید که مثلاً مشتریان کلاس ۱ را از کلاس ۳ جدا کند.
2. **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. الگوها بسیار پیچیده‌تر و **غیرخطی** هستند.
3. **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 (بلوک 4.1.4 Hexbin)**

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

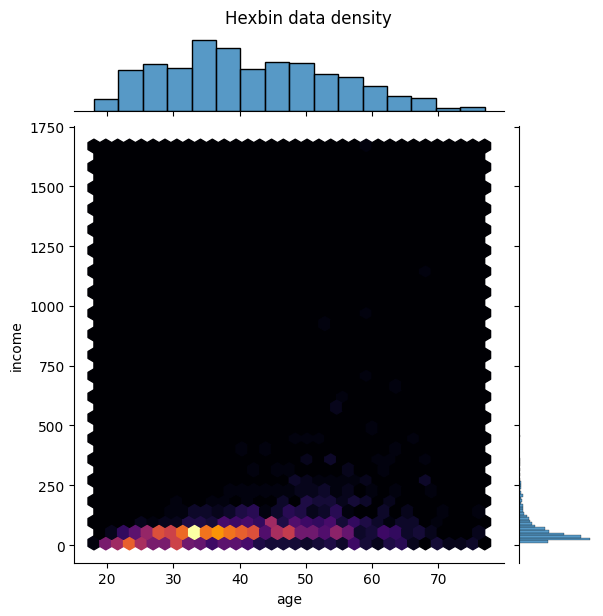
هدف این کد، حل این مشکل با استفاده از jointplot(kind='hex') است. این نمودار، فضای دوبعدی را به شش‌ضلعی‌های کوچک تقسیم می‌کند و هر شش‌ضلعی را بر اساس تعداد نقاط داده‌ای که در آن قرار دارند، رنگ‌آمیزی می‌کند.

**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 (مثلاً در سنین ۳۰ تا ۵۰ سال و درآمدهای پایین‌تر) نشان‌دهنده تجمع اصلی مشتریان است.

نتیجه کد:



این نمودار از ۳ بخش تشکیل شده:

1. نمودار مرکزی (شش‌ضلعی‌ها):
   * 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.
2. 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 یافته‌ی نمودار مرکزی را تأیید می‌کند.
3. 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.

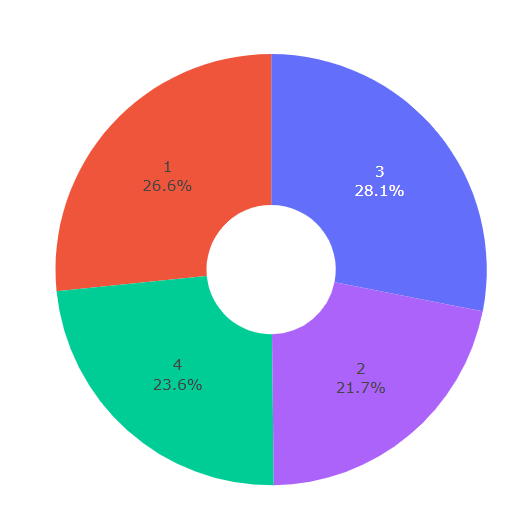
**توضیح کد (بلوک 4.1.4 Pieplot)**

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**

* + ابتدا ، 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 (index, custcat) را به نام‌های معنادار (custcat, count) تغییر می‌دهیم. و سپس مقادیر عددی ۱, ۲, ۳, ۴ را به مقادیر متنی "1", "2", "3", "4" 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 **۲۸.۱٪**، بزرگترین بخش است.
* **کلاس ۱ (Basic Service):** با **۲۶.۶٪**، در رتبه دوم قرار دارد.
* **کلاس ۴ (Total Service):** با **۲۳.۸٪**، در رتبه سوم است.
* **کلاس ۲ (E-Service):** با **۲۱.۵٪**، کوچکترین بخش است.

داده‌ها متعادل هستند: بزرگترین کلاس (۲۸.۱٪) و کوچکترین کلاس (۲۱.۵٪) تفاوت زیادی با هم ندارند

**توضیح کد (بلوک 4.1.4 Countplot)**

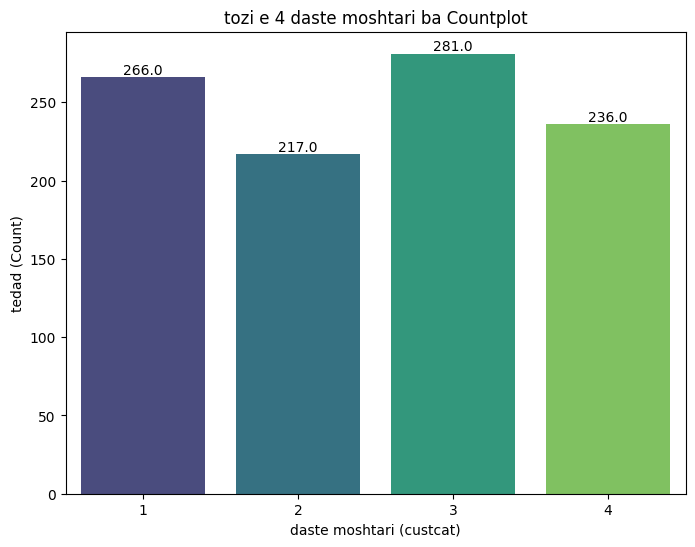
هدف این کد دقیقاً مشابه نمودار دایره‌ای (Pie Plot) قبلی است که همان شمارش تعداد نمونه‌ها در هر یک از ۴ دسته مشتری. با این تفاوت که خواناتر و راحت تر است

**عملکرد کد**

1. **ax = sns.countplot(...)**:
   * از تابع countplot کتابخانه seaborn استفاده می‌کنیم.
   * **x='custcat'**: شما به seaborn می‌گویید که ستون custcat را روی محور X قرار دهد.
   * countplot به طور خودکار داده‌ها را **گروه‌بندی** می‌کند، **تعداد** هر گروه را می‌شمارد و یک نمودار میله‌ای (Bar Plot) از آن ترسیم می‌کند که در آن ارتفاع هر میله، همان شمارش (Count) است.
   * ax.patches لیستی از میله‌ها روی نمودار می‌دهد.
   * ax.annotate(...): این دستور، متن (f'{p.get\_height()}' که همان ارتفاع یا شمارش میله است) را می‌گیرد و آن را دقیقاً در بالای هر میله ((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**

1. **مقایسه آسان:** این نمودار (نسبت به نمودار دایره‌ای) مقایسه را آسان‌تر می‌کند. با یک نگاه می‌توانید ببینید که ارتفاع میله‌ها به هم نزدیک است.
2. **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 مستقیما به یک مدل مانند رگرسیون لجستیک بدهیم، مدل به اشتباه فکر می‌کند که region 3 از region 1 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:**

1. **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.
2. **X\_features\_df = df.drop('custcat', axis=1)**:
   * یک DataFrame جدید (X\_features\_df) می‌سازد که فقط شامل **ویژگی‌ها (X)** است.
3. **categorical\_cols = [...]**:
   * شما لیستی از تمام ستون‌هایی که در گام قبلی به عنوان طبقه‌ای شناسایی شده است را تعریف می‌کنید.
   * **pd.get\_dummies**:
     + این تابع ستون‌های عددی (مثل age, income, tenure) را که در لیست categorical\_cols نیستند، دست‌نخورده باقی می‌گذارد سپس هر ستون در categorical\_cols را می‌گیرد و آن را به ستون‌های جدید (مثلاً region\_1, region\_2, region\_3) تبدیل می‌کند.

**نتیجه کد:**

1. **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.
2. **تغییر نوع داده (Dtype):**
   * ۵ ستون عددی اصلی (tenure, age, address, income, employ) دست‌نخورده باقی مانده‌اند (همان int64 و float64).
   * ۲۲ ستون جدید (مانند region\_1, marital\_0, ed\_1 و...) همگی از نوع **bool** هستند که pandas از آن به جای ۰ و ۱ استفاده می‌کند. این دقیقاً همان چیزی است که ما می‌خواستیم.

**نرمال سازی داده ها 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) می‌آوریم.

**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) یک آرایه NumPy برمی‌گردانند. 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)**:
  + fit: ماشین Scaler، حداقل (Min) و حداکثر (Max) مقدار را برای *هر ۲۷ ستون* پیدا می‌کند.
  + transform: تمام مقادیر را بر اساس Min و Max خودشان، به بازه ۰ تا ۱ تبدیل می‌کند.
  + این کار برای ستون‌های One-Hot (مثل region\_1 که Min=0 و Max=1 است) مشکلی ایجاد نمی‌کند؛ آن‌ها دست‌نخورده (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 می شود.

نتیجه کد:

* + همانطور که می‌بینید، **تمام ۲۷ ستون** شما (هم ستون‌های عددی اصلی و هم ستون‌های One-Hot جدید) اکنون دارای min برابر 0.0 و max برابر 1.0 هستند.

شما با موفقیت یک دیتاست (X\_normalized\_df) ایجاد کرده‌اید که:

**حذف ویژگی های غیر مفید یا تکراری:**

**حذف ردیف‌های تکراری:**

* **num\_duplicates = X\_normalized\_df.duplicated().sum()**: این دستور تعداد ردیف‌هایی را که یک کپی دقیق از ردیف‌های قبلی 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]**: این یک گام **بسیار مهم** است. این کد تضمین می‌کند که اگر ردیفی 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 (region\_1, marital\_0, ...)**

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

1. **قبل از انکودینگ:** شما یک ستون gender با مقادیر ۰ و ۱ داشتید.
2. **بعد از انکودینگ (get\_dummies):** شما **دو** ستون جدید ایجاد کردید: gender\_0 و gender\_1.

* اگر gender\_0 برابر 1 باشد، gender\_1 **الزاماً** 0 است.
* اگر gender\_0 برابر 0 باشد، gender\_1 **الزاماً** 1 است.

این یعنی دو ستون gender\_0 و gender\_1 **اطلاعات کاملاً یکسانی** را حمل می‌کنند؛ آن‌ها به شدت به هم وابسته‌اند

نتیجه‌گیری:

region\_1، marital\_0، ed\_1، retire\_0.0، gender\_0 و reside\_1 را حذف شد به این دلیل که اضافه بودند. این کار، دیتاست 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 کرده‌ایم، ارزیابی کنیم. پس ما کل داده‌های پاک‌سازی شده (X\_final و y\_final) را به دو بخش کاملاً مجزا تقسیم می‌کنیم:

1. **داده‌های آموزشی (Train Set):** (۸۰٪ داده‌ها) مدل، الگوها را **فقط** از این بخش یاد می‌گیرد.
2. **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:**

1. **X\_train, X\_test, y\_train, y\_test = train\_test\_split(...)**:
   * این تابع اصلی از کتابخانه scikit-learn است که عمل تقسیم‌بندی را انجام می‌دهد و ۴ خروجی مجزا برمی‌گرداند.
2. **X\_final, y\_final**:
   * این‌ها ورودی‌ها هستند: دیتاست ۲۱ ستونی که کاملاً پاک‌سازی، انکود و نرمال شده است (X\_final) و برچسب‌های هدف مربوط به آن (y\_final).
3. **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.
4. **random\_state=42**:
   * این یکی از مهم‌ترین پارامترها برای تکرارپذیری (Reproducibility) است.
   * عمل تقسیم‌بندی به صورت "تصادفی" انجام می‌شود. اگر این پارامتر را نگذارید، هر بار که کد را اجرا می‌کنید، ردیف‌های متفاوتی در مجموعه آموزشی و آزمایشی قرار می‌گیرند و نتایج مدل شما (مثلاً دقت) در هر اجرا تغییر می‌کند.
   * با تنظیم random\_state روی یک عدد ثابت این "تصادفی بودن" همیشه به یک شکل اتفاق میفتد. این کار اجازه می‌دهد که نتایج را دقیقاً بازتولید کنند.

نتیجه کد:

--- گام اول: در حال تقسیم داده‌ها به آموزشی (80%) و آزمایشی (20%) ---

تقسیم‌بندی با موفقیت انجام شد.

ابعاد X\_train (آموزشی): (800, 21)

ابعاد X\_test (آزمایشی): (200, 21)

**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 (Lasso و RFE) 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.
* **alpha=0.01:** این پارامتر، قدرت جریمه را مشخص می‌کند. alpha=0.01 یک جریمه نسبتاً ملایم است که به مدل اجازه می‌دهد تعدادی از ویژگی‌های خوب را نگه دارد.
* **selected\_features\_lasso = ... [lasso\_coeffs != 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.
* **base\_model = LogisticRegression(...)**: 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.
* **n\_features\_to\_select=10**: به RFE یک دستور داده می شود " دقیقاً ۱۰ ویژگی برتر احتیاج است."
* **rfe.fit(X\_train, y\_train)**: این دستور، یک فرآیند تکراری را آغاز می‌کند:
  1. مدل LogisticRegression را با **تمام ۲۱** ویژگی آموزش می‌دهد.
  2. **ضعیف‌ترین** ویژگی را پیدا می‌کند و آن را **حذف** می‌کند.
  3. دوباره مدل را با **۲۰** ویژگی باقی‌مانده آموزش می‌دهد.
  4. ضعیف‌ترین ویژگی را پیدا و حذف می‌کند.
  5. ... این فرآیند را آنقدر تکرار می‌کند تا **فقط ۱۰ ویژگی برتر** باقی بمانند.

نتیجه کد:

========================================

--- روش ۱: انتخاب ویژگی با Lasso ---

Lasso تعداد 10 ویژگی را انتخاب کرد:

['tenure', 'employ', 'region\_2', 'marital\_1', 'ed\_2', 'ed\_3', 'ed\_4', 'ed\_5', 'gender\_1', 'reside\_6']

========================================

--- روش ۲: انتخاب ویژگی با RFE ---

RFE تعداد 10 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\_lasso = [...] و selected\_features\_rfe = [...]**:
  + 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 Lasso 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 تکرار می‌شود).

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

* **مدل ۱ (Lasso):**
  + log\_reg\_lasso = LogisticRegression(...): یک مدل رگرسیون لجستیک جدید می‌سازد.
  + log\_reg\_lasso.fit(X\_train\_lasso, y\_train): مدل را **فقط** با ۱۰ ویژگی Lasso آموزش می‌دهد.
  + acc\_lasso = accuracy\_score(...): 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 (پایه - Baseline):**
  + این **مهم‌ترین بخش** آزمایش شماست.
  + log\_reg\_all = LogisticRegression(...): یک مدل سوم می‌سازد.
  + log\_reg\_all.fit(X\_train, y\_train): این مدل را با داده‌های آموزشی **اصلی** (شامل **تمام ۲۱ ویژگی** پاک‌سازی شده) آموزش می‌دهد.
  + acc\_all = accuracy\_score(...): دقت این مدل "پایه" را محاسبه می‌کند.

**۳. مقایسه نهایی**

با مقایسه acc\_lasso و acc\_rfe با acc\_all، می‌توان به طور قطعی گفت که آیا فرآیند **انتخاب ویژگی** (Feature Selection) مفید بوده است یا خیر.

* **اگر acc\_lasso > acc\_all**: یعنی Lasso با موفقیت "نویز" را حذف کرده و مدلی بهتر با ویژگی‌های کمتر ساخته است.
* **اگر acc\_lasso < acc\_all**: یعنی Lasso ویژگی‌های مفیدی را حذف کرده است.
* **اگر acc\_lasso ≈ acc\_all**: 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.

تعداد ویژگی‌های Lasso: 10

تعداد ویژگی‌های RFE: 10

========================================

--- مدل ۱: آموزش با ۱۰ ویژگی منتخب Lasso ---

✅ دقت (ویژگی‌های Lasso): 38.50%

========================================

--- مدل ۲: آموزش با ۱۰ ویژگی منتخب RFE ---

✅ دقت (ویژگی‌های RFE): 38.50%

========================================

--- مدل ۳: مدل پایه (تمام ۲۱ ویژگی پاک‌سازی شده) ---

✅ دقت (تمام ۲۱ ویژگی): 37.00%

==================================================

--- 📊 مقایسه نهایی دقت مدل‌ها ---

مدل پایه (۲۱ ویژگی): 37.00%

مدل RFE (۱۰ ویژگی): 38.50%

مدل Lasso (۱۰ ویژگی): 38.50%

(در این مورد، نتایج نشان داد که هر دو مدل Lasso و RFE (با دقت ۳8.5٪) از مدل پایه (با دقت ۳8.5٪) **بهتر** عمل کردند،)

**ماتریس در هم ریختگی و نمودار ROC و مقدار AUC:**

**۱. ماتریس درهم‌ریختگی (Confusion Matrix)**

* + **اعداد روی قطر اصلی (از بالا-چپ 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 خطاها 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 تفکیک‌کننده 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 (مثلاً) 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 (coef\_) 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.
* **راه‌حل:**
  + **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 تمیز 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 ---

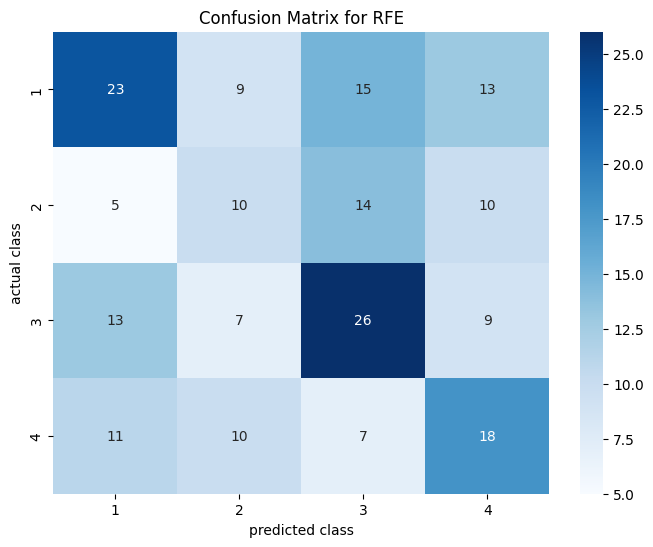
امتیازهای AUC (سطح زیر نمودار) برای هر کلاس:

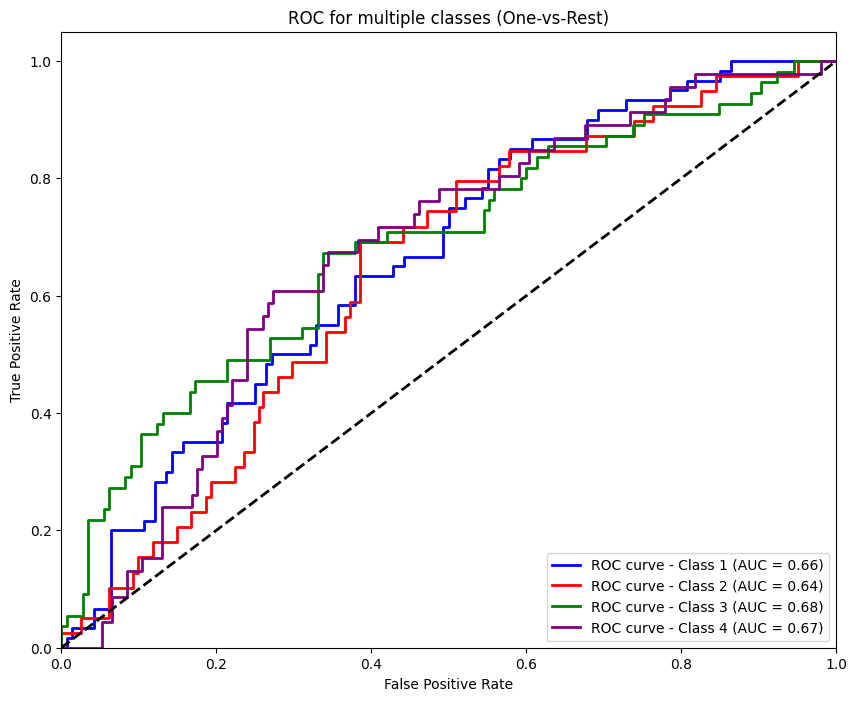
- AUC برای کلاس 1 (در برابر بقیه): 0.66

- AUC برای کلاس 2 (در برابر بقیه): 0.64

- AUC برای کلاس 3 (در برابر بقیه): 0.68

- AUC برای کلاس 4 (در برابر بقیه): 0.67





## -4-4

**کاهش ابعاد (PCA و LDA)**

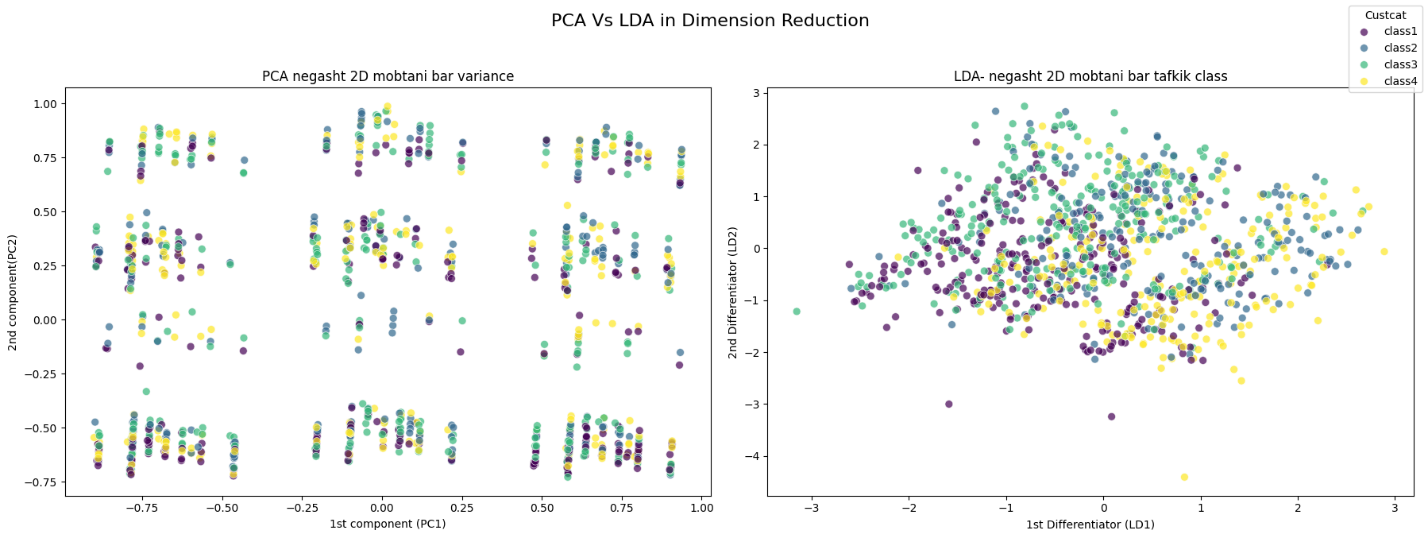
**روش اول - PCA (تحلیل مؤلفه‌های اصلی)**

* **هدف:** کاهش ابعاد با استفاده از روش **(Unsupervised)**.
* **pca = PCA(n\_components=2)**: یک مدل PCA می‌سازد و به آن می‌گوید که می‌خواهد داده‌ها را به **۲ مؤلفه** کاهش دهد.
* **X\_pca = pca.fit\_transform(X\_final)**:
  + fit: PCA داده‌های X\_final را تحلیل می‌کند تا ۲ محوری را پیدا کند که **بیشترین واریانس (پراکندگی)** داده‌ها را توضیح می‌دهند. PCA در این مرحله **اصلاً به y\_final (برچسب کلاس‌ها) نگاه نمی‌کند**.
  + transform: داده‌های ۲۱ بُعدی را روی آن ۲ محور جدید (Project) می‌کند.
* **pca\_df = ...**: 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)**.
* **lda = LinearDiscriminantAnalysis(n\_components=2)**: یک مدل LDA می‌سازد که هدفش کاهش به **۲ بُعد** است.
* **X\_lda = lda.fit\_transform(X\_final, y\_final)**:
  + **تفاوت با PCA:** این تابع هم X\_final (ویژگی‌ها) و هم **y\_final (برچسب‌ها)** را به عنوان ورودی می‌گیرد.
  + **fit**: LDA به دنبال محورهایی نمی‌گردد که بیشترین واریانس را دارند؛ بلکه به دنبال محورهایی می‌گردد که بهترین تفکیک (جداسازی) را بین ۴ کلاس مشتری ایجاد می‌کنند.
* **lda\_df = ...**: 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 خوشه 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 بُعدی شما، برخلاف چیزی که PCA نشان داد، به خوبی قابل تفکیک خطی هستند. LDA با موفقیت توانسته داده‌ها را طوری در دو بُعد 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 pandas (که X\_train.values هستند) را به فرمت بومی PyTorch به نام 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 (.fit())، DataLoader داده‌ها را به "دسته‌های" (batches) ۳۲ تایی می‌شکند.
  + **shuffle=True**: قبل از هر دور آموزش (epoch)، داده‌ها را کاملاً بُر می‌زند. 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، ۴ کلاس نهایی را پیش‌بینی کند.

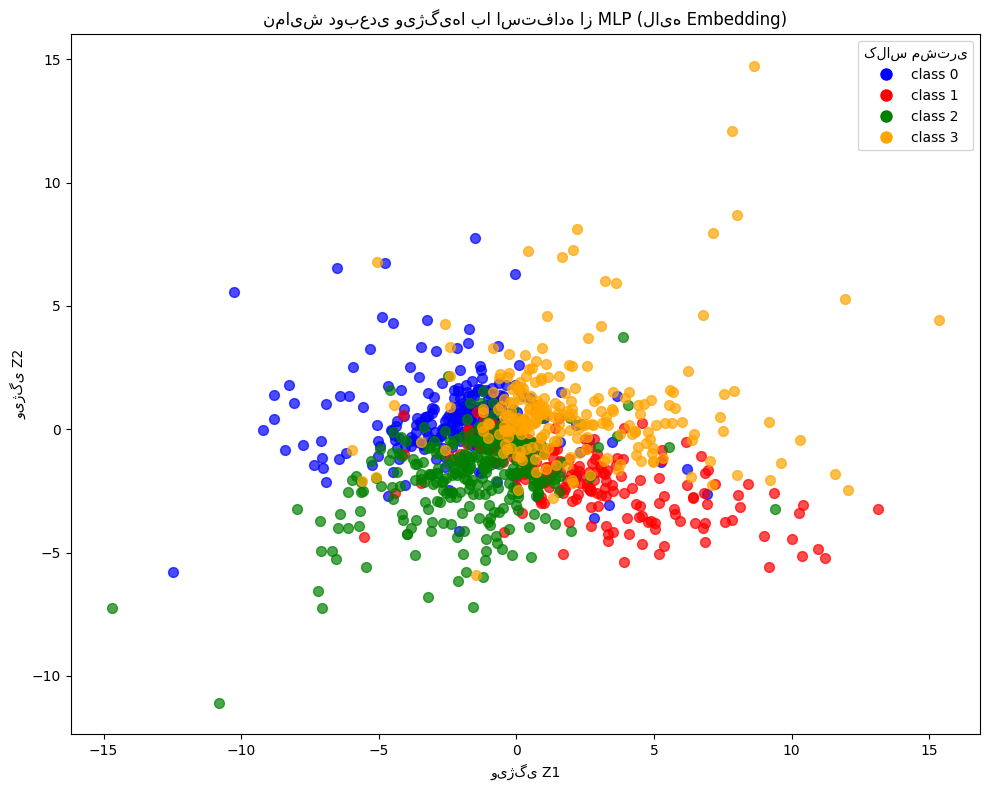
**تابع forward(self, x):**

این تابع، جریان داده در شبکه را تعریف می‌کند:

1. داده‌ها (x) از لایه‌های fc1, fc2, fc3 عبور می‌کنند.
2. **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.
3. **y = self.out(z)**: خروجی z به لایه نهایی داده می‌شود تا پیش‌بینی ۴ کلاسه (y) تولید شود.
4. **return y, z**: مدل به طور همزمان **هم** پیش‌بینی نهایی (y، برای محاسبه خطا) و **هم** نمایش ۲ بُعدی (z، برای رسم نمودار) را برمی‌گرداند.

**۳. آموزش و استخراج**

* + **loss.backward()** و **optimizer.step()**: این دو دستور، فرآیند "یادگیری" واقعی هستند (Backpropagation). مدل خطا 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.
* **model.eval() و with torch.no\_grad()**:
  + 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 (y, z) را برمی‌گرداند، کد y (پیش‌بینی) را نادیده می‌گیرد (\_) و z (مقادیر ۲ بُعدی) را در Z\_train ذخیره می‌کند.
* **Z\_combined = ... و plt.scatter(...)**:
  + در نهایت، داده‌های ۲ بُعدی 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 بُعدی داده‌ها را که توسط "مغز" شبکه عصبی (لایه fc4\_embedding) آموخته شده است، نشان می‌دهد.

1. **جداسازی غیرخطی :**
   * برخلاف PCA (که کاملاً درهم‌ریخته بود) و LDA (که خوشه‌های خطی و کمی هم‌پوشانی داشت)، این نمودار، کلاس‌ها را به **اشکال پیچیده و غیرخطی** تبدیل کرده است.
   * به وضوح می‌توانید ببینید که کلاس‌ها (مخصوصاً 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 بُعدی (z)، داده‌ها را طوری "تغییر شکل" دهد (Warp) که کاملاً از هم قابل تفکیک باشند.
   * این نمودار، آن "فضای تغییرشکل‌یافته" را به ما نشان می‌دهد.

**مقایسه نهایی با LDA و PCA**

* **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.

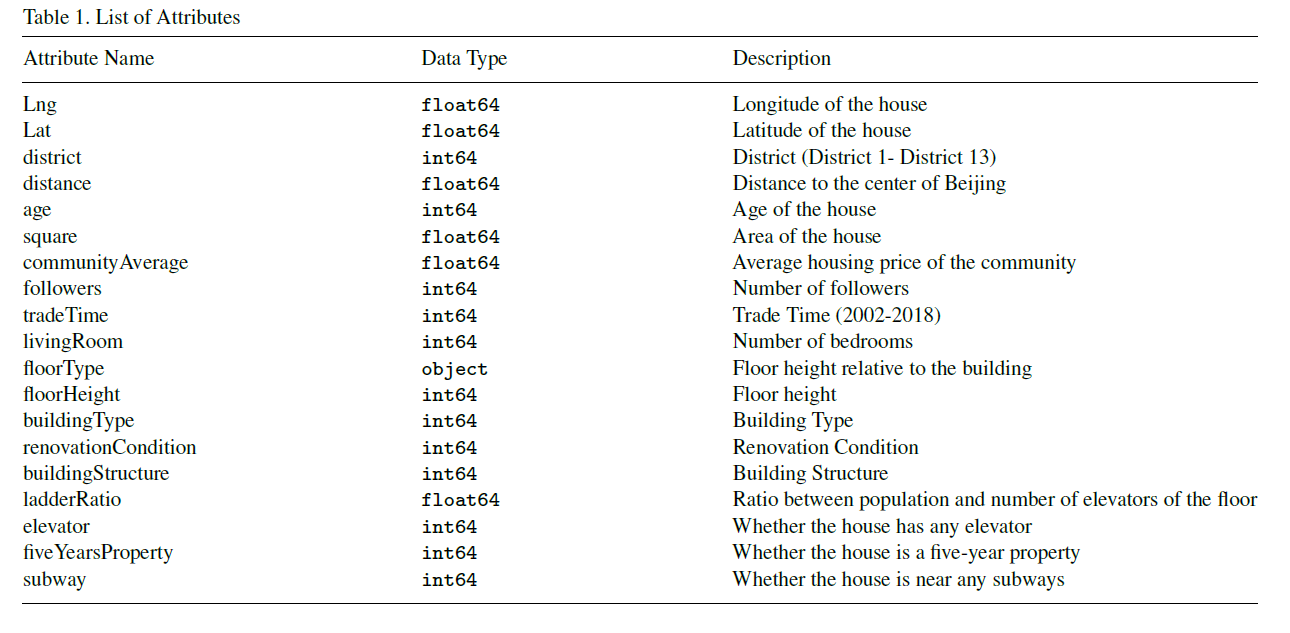
# **سوال 5**

[کد سوال](https://colab.research.google.com/drive/1ugABFcK--Sb9DqOMw7G1LPjA2bbnlmkc?usp=sharing)

## **بخش اول:شهر بیجینگ**

دیتا ست Housing Price in Beijing شامل ۳۰۰هزار رکورد از معاملات ۲۰۰۹ تا ۲۰۱۸ با ۲۶ متغیر اولیه؛ پس از پاک‌سازی ۲۳۱٬۹۶۲ نمونه و **۱۹ ویژگی** باقی مانده‌اند

19 ویژگی:



1حذف داده‌های ناقص و ویژگی‌های بی‌کاربرد: متغیرهایی که **بیش از ۵۰٪ دادهٔ گمشده** داشتند، حذف شدند.متغیر

Dayonmarket به دلیل داشتن ۱۵۷٬۹۷۷ مقدار گمشده حذف شد. ردیف‌هایی که هر نوع دادهٔ ناقص داشتند نیز حذف شدند

2چندین تغییر برای پاک‌سازی و بهبود داده انجام شد: ذف ویژگی‌های مربوط به تعداد **آشپزخانه، حمام و اتاق پذیرایی** به‌دلیل ابهام در معنی.living room به معنای تعداد اتاق خوای یوده که به 1 تا 4 محدود شده. فزودن ویژگی جدید **distance** که فاصلهٔ هر خانه از مرکز پکن را نشان می‌دهدجایگزینی ویژگی constructionTime با age (سن بنا = سال جاری − سال ساخت). تعیین مقادیر حداقل برای **قیمت** و **مساحت** تا داده‌های غیرواقعی حذف شوندتقسیم ویژگی **floor** به دو ویژگی جداگانه: **floorType** (نوع طبقه) و **floorHeight** (ارتفاع طبقه

3از **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

4در پایان، داده‌ها به دو بخش **آموزش و آزمون** با نسبت **۴ به ۱** تقسیم شدند

پ) Linear RegressionوRandom ForestوExtreme Gradient Boosting (XGBoost)وLight Gradient Boosting Machine (LightGBM)وHybrid Regression ModelوStacked Generalization

بخش دوم)

1.تعداد داده ها و ویژگی ها :تعداد نمونه 545 وتعداد ویژگی 13 تا

2.

|  |  |
| --- | --- |
| price | عددی (int) |

|  |  |
| --- | --- |
| 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) |

3

|  |  |
| --- | --- |
| 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، الگوها، 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.

2

A screen shot of a black screen

AI-generated content may be incorrect.

**۶ ویژگی عددی** دارد:  
['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']

**۷ ویژگی دسته‌ای (متنی)** دارد:  
['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']

3.

A graph of a number of blue squares

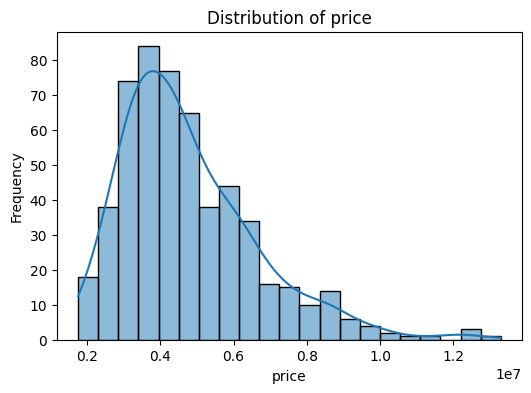
AI-generated content may be incorrect.

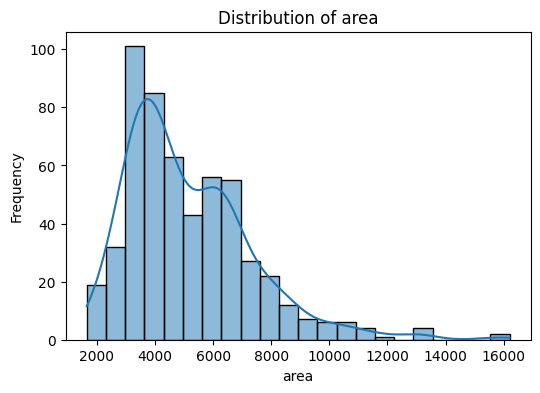
3A graph of a number of blue squares

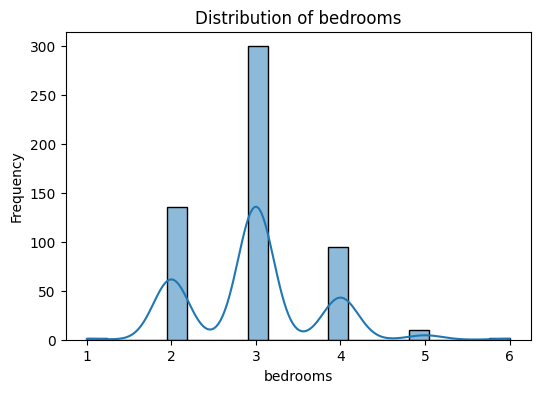
AI-generated content may be incorrect.

A graph of a bar

AI-generated content may be incorrect.







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 (Outliers)** است.
* ویژگی **bedrooms** توزیع محدودی دارد (بین ۱ تا ۶) و داده پرت خاصی مشاهده نمی‌شود.

6.



بخش چهارم)پیش پردازش

1. هیچ سطر تکراری وجود ندارد بنابراین نیاز به حذف نیست
2. در هیچ‌یک از ستون‌ها مقدار گمشده‌ای وجود ندارد (همهٔ شمارنده‌ها ۰ هستند).

اگر missing داشتیم عددی ها → میانه (Median): در حضور داده‌های پرت، از میانگین مقاوم‌تر است.

دسته ای ها→ پرتکرارترین مقدار (Mode): ساختار طبقه‌بندی را حفظ می‌کند و از ایجاد دستهٔ جدید جلوگیری می‌کند.

1. **Label Encoding**

هر مقدار متنی را به یک عدد اختصاص می‌دهد.

مثلاً:

| **وضعیت مبلمان** | **کد** |
| --- | --- |
| 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.

مثلاً:

| **وضعیت مبلمان** | **furnished** | **semi-furnished** | **unfurnished** |
| --- | --- | --- | --- |
| 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.

1. **Ordinal Encoding**

اگر داده‌ها ترتیب منطقی دارند (مثلا “low”<”Medium”<”High”)می‌توان به آن‌ها عدد متناظر داد:

|  |  |
| --- | --- |
| low | 1 |
| Medium | 2 |
| High | 3 |

کد کیفیت:

🔸**مناسب برای:** ویژگی‌های **دارای ترتیب**.

**4. Target Encoding / Mean Encoding**

در این روش، هر مقدار متنی با میانگین مقدار هدف (مثل قیمت) برای آن مقدار جایگزین می‌شود.

🔸 **مناسب برای:** داده‌های بزرگ و ویژگی‌های با مقادیر زیاد (مثلاً نام محله‌ها در پیش‌بینی قیمت).

در دیتاست ما، تمام ویژگی‌های دسته‌ای از نوع **دوبخشی (yes/no)** یا دارای چند دسته‌ی محدود هستند؛ بنابراین:

✅ **بهترین انتخاب:** **One-Hot Encoding**  
زیرا هیچ ترتیب طبیعی بین مقادیر وجود ندارد (مثلاً “furnished” از “unfurnished” بزرگ‌تر نیست).

**4.**

**روش‌های متداول:**

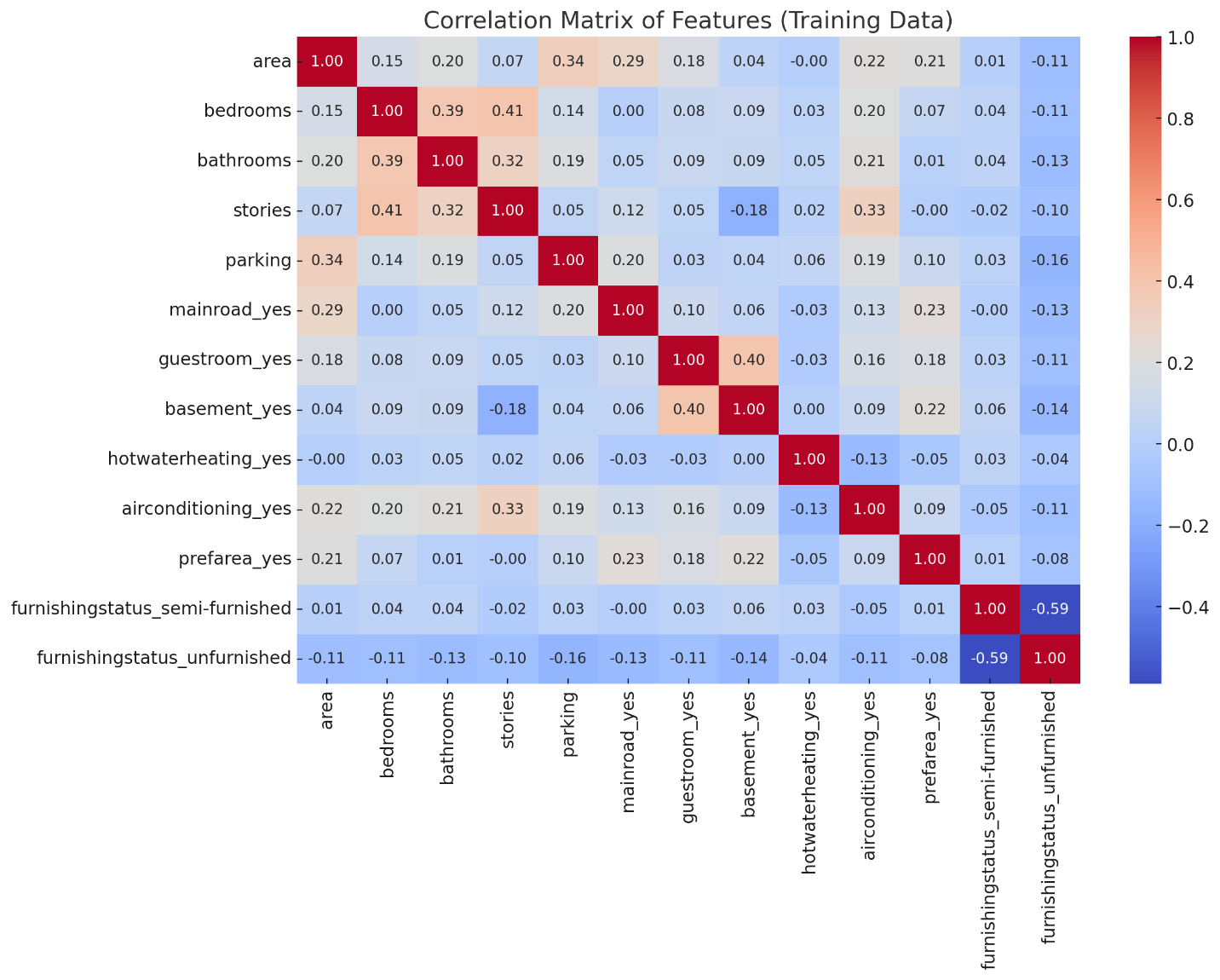
| **روش** | **توضیح** | **کاربرد** |
| --- | --- | --- |
| **۱. روش آماری (Z-Score)** | داده‌هایی که بیش از ۳ انحراف معیار از میانگین فاصله دارند پرت محسوب می‌شوند. | وقتی داده‌ها توزیع نرمال دارند. |
| **۲. روش IQR (Interquartile Range)** | داده‌هایی خارج از بازه‌ی [Q1 - 1.5×IQR, Q3 + 1.5×IQR] پرت هستند. | برای داده‌های غیرنرمال؛ پرکاربردتر در عمل. |
| **۳. روش بصری** | با نمودار جعبه‌ای (Boxplot) یا پراکندگی (Scatter) تشخیص داده می‌شود. | بررسی بصری قبل از حذف. |
| **۴. روش مدل‌محور** | استفاده از مدل‌هایی مثل Isolation Forest یا DBSCAN برای شناسایی نقاط ناهنجار. | وقتی داده پیچیده و چندبعدی است. |

نتیجه حذف داده‌های پرت با روش **IQR**

* شکل اولیهٔ داده‌ها: **(545, 13)**
* پس از حذف پرت‌ها: **(365, 13)**
* تعداد ردیف‌های حذف‌شده: **180**

یعنی حدود **۳۳٪ از داده‌ها** دارای مقادیر پرت در یکی از ویژگی‌های عددی بودند و حذف شدند.

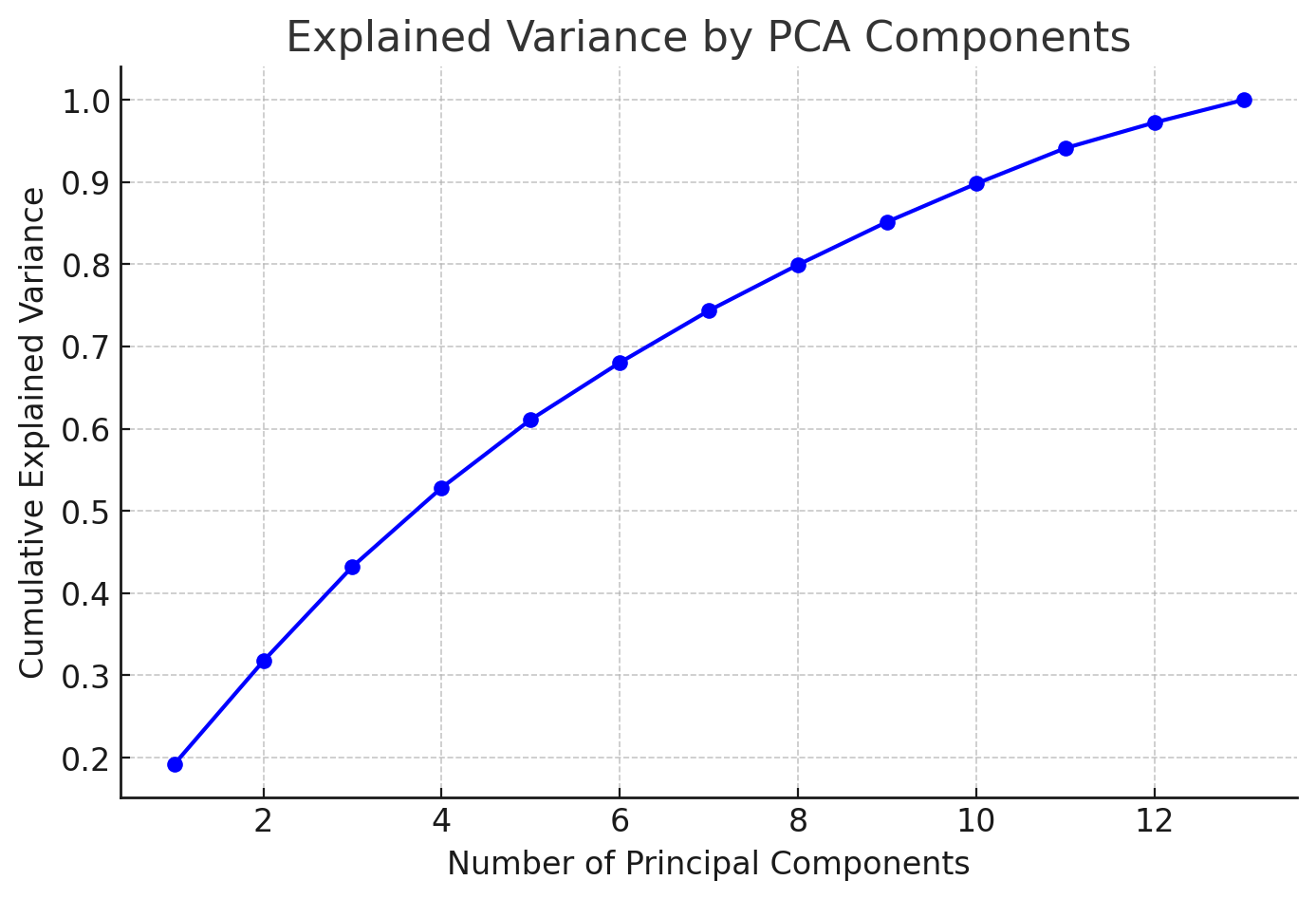
بخش پنجم) انتخواب وبژگی



 نمودار حرارتی (Heatmap) میزان ارتباط بین تمام ویژگی‌ها را نشان می‌دهد.

 قوی‌ترین همبستگی منفی مشاهده‌شده بین دو ویژگی زیر است:

* furnishingstatus\_semi-furnished و furnishingstatus\_unfurnished
* مقدار همبستگی ≈ **−0.59**

2. 

* برای اینکه مدل بتواند **حدود ۹۵٪ از واریانس کل داده** را حفظ کند، باید **۱۲ مؤلفهٔ اصلی (Principal Components)** در نظر بگیریم.این یعنی نمیتوانیم به طور جدی هیچ وبژگی 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.
* **✳️ علائم وجود هم‌خطی:**
* ضرایب مدل ناپایدار می‌شوند.
* ضریب تعیین (R²) بالا، ولی ضرایب آماری (p-value) غیرمعنادار.
* مقادیر **VIF (Variance Inflation Factor)** بالا.
* **🔹 بخش دوم: روش‌های بررسی و رفع هم‌خطی**

| * **روش** | * **توضیح** | * **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 به‌صورت بازگشتی حذف می‌کند تا بهترین مجموعه باقی بماند. | * انتخاب زیرمجموعه‌ی بهینه از ویژگی‌ها برای مدل |

* **🔹 بخش سوم: پیاده‌سازی — بررسی VIF (عامل تورم واریانس)**
* 🔹 **نتایج شاخص تورم واریانس (VIF):**

| * **ویژگی** | * **مقدار VIF** |
| --- | --- |
| * 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:**
* معمولاً اگر **VIF > 10** باشد، متغیر دارای هم‌خطی شدید است و باید حذف شود.
* در این داده‌ها همهٔ مقادیر **VIF < 2** هستند، یعنی **هیچ هم‌خطی چندگانه قابل‌توجهی 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 بازگشتی ویژگی‌ها):**
* مدل **رگرسیون خطی** پس از بررسی تمام ویژگی‌ها،  
  این **۸ ویژگی برتر** را به‌عنوان مؤثرترین متغیرها در پیش‌بینی قیمت انتخاب کرده است:

| * **ویژگی منتخب** | * **توضیح** |
| --- | --- |
| * area | * بزرگ‌تر بودن مساحت معمولاً قیمت را افزایش می‌دهد |
| * bathrooms | * تعداد بیشتر سرویس‌ها معمولاً نشانهٔ خانه‌های لوکس‌تر است |
| * stories | * تعداد طبقات نشان‌دهندهٔ 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 | * تهویه مطبوع معمولاً در خانه‌های گران‌تر دیده می‌شود |
| * prefarea\_yes | * قرار داشتن در منطقهٔ 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** به‌صورت خودکار ویژگی‌های غیرمؤثر را حذف کرد.
* در نهایت ۸ ویژگی بالا را به‌عنوان **مؤثرترین ویژگی‌ها برای مدل‌سازی** انتخاب کردیم.

بخش ششم)

نتایج ارزیابی ۴ رگرسور خواسته‌شده (به‌علاوه‌ی MLP) روی همین داده‌مان 👇

| **مدل** | **MAE** | **RMSE** | **R² (Test)** | **R² (Train)** |
| --- | --- | --- | --- | --- |
| **Lasso** | 704,226 | 1,036,279 | **0.497** | 0.571 |
| **Multiple Linear** | 704,466 | 1,036,438 | **0.497** | 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 **Lasso** و **Linear Regression** تقریباً برابر (R²≈0.497) داشتند.
* **Ridge** کمی ضعیف‌تر شد؛ احتمالاً چون تنظیم L2 ضرایب را بیش از حد کوچک کرده است.
* **Polynomial درجه ۲** با اینکه روی Train خیلی خوب جا افتاده (R²=0.738)، روی Test افت کرده (R²=0.400) ⇒ **بیش‌برازش**.
* **MLP** در این تنظیمات خوب جواب نداده (R² منفی)؛ برای دادهٔ کم‌حجم و ویژگی‌های محدود، شبکه عمیق بدون تنظیمات دقیق معمولاً بیش‌برازش یا ناپایداری می‌دهد.

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

* **L1 (Lasso):** برخی ضرایب را دقیقاً صفر می‌کند → **انتخاب ویژگی**.
* **L2 (Ridge):** ضرایب را کوچک می‌کند → **کاهش واریانس** و مقابله با چندخطی.

دو پارامتر کلیدی دارد:

* alpha (λ) شدت کلی منظم‌سازی را کنترل می‌کند (هرچه بزرگ‌تر، منظم‌سازی قوی‌تر).
* l1\_ratio (0..1) نسبت ترکیب L1/L2 را مشخص می‌کند:
  + نزدیک 0 ⇒ شبیه **Ridge**
  + نزدیک 1 ⇒ شبیه **Lasso**
  + مقدار میانی ⇒ **Elastic Net** (ترکیبی)

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

**Elastic Net** بهترینِ هر دو دنیا را می‌گیرد: هم انتخاب ویژگی، هم پایداری در حضور هم‌خطی.

7)استفاده از mlp تحت انتخاب کننده ویژگی

1. قیمت را با **چارک‌ها** به ۴ کلاس تبدیل کردیم (۴-کلاسه شدن مسئله).
2. پیش‌پردازش قبلی (حذف پرت‌ها با IQR، وان‌هات، استانداردسازی) انجام شد.
3. یک **MLPClassifier** با ساختار (64, 32) و خروجی ۴ نورون (Softmax) آموزش دادیم.
4. با استفاده از وزن‌ها و بایاس‌های شبکه (‌coefs\_, intercepts\_) یک **forward pass** نوشتیم تا **اکتیواسیونِ لایهٔ پنهان آخر** را به‌عنوان «ویژگی‌های آموخته‌شده» استخراج کنیم.
5. همان خانواده‌ مدل‌ها را دو بار آموزش دادیم و سنجیدیم:
   * **Baseline** روی ویژگی‌های اولیه
   * **مدل‌ها روی ویژگی‌های استخراج‌شدهٔ MLP**

**نتایج (دقت و F1 ماکرو)**

**الف) روی ویژگی‌های اولیه (Baseline):**

* 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 استخراج‌شده 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 قابل‌مشاهده** ایجاد کرد:  
  بهترین baseline (Poly+LogReg) دقت ≈ **0.49** داشت؛ بهترین مدل با ویژگی‌های MLP (RidgeClassifier) به **0.52** رسید.
* چرا؟
  + لایهٔ 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 می‌کنند.
* چرا Poly(2) روی ویژگی‌های MLP بدتر شد؟
  + چون خود MLP قبلاً نگاشت غیرخطی را یاد گرفته؛ افزودن چندجمله‌ای درجه‌۲ روی آن **بیش‌برازش** و شلوغی غیرضروری می‌آورد.
* تفاوت «L1» و «L2»:
  + روی ویژگی‌های MLP، **Ridge** از Logistic-L1 بهتر بود؛ در فضای تعبیهٔ (embedding) کوچک، **پایدارسازی L2** از صفرسازی L1 مؤثرتر است.