به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم های هوشمند

گزارش پروژه

مینی پروژه شماره 1

[محمدپارسا زارع 40118953]

[محسن کریمی 40121913]

استاد : آقای دکتر علیاری

آبان 1404

پرسش های تحلیلی

پرسش یک)

الف :

$$Sensitivity = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP}$$

ب)

Specificity	Sensitivity	TN	FN	FP	TP	کلاس
94.74%	88.24%	90	6	5	45	C1
93.20%	74.42%	96	11	7	32	C2
96.55%	53.33%	112	14	4	16	C3
86.29%	90.91%	107	2	17	20	C4

پرسش دوم)

الف)

۲- پرسش دوم

الف)

$\underline{w}^{*T}\underline{x}^{sv}+b=1$

$w^{*}=\sum_{sv}\alpha_i\,\underline{x}^{sv}y^{i}$

$\tilde{x}=[x_1,x_2,1]^T$, $\tilde{w}=[w_1,w_2,b]^T$

$y=+1\rightarrow A^{+}$, $y=-1\rightarrow A^{-}$

۱) $(1,1)\in A^{+}$ $\tilde{x}_1=(1,1,1)$, $y_1=+1$, $\tilde{w}(0)^T\tilde{x}_1=0$

$\tilde{w}(1)=\tilde{w}(0)+y_1\tilde{x}_1=(0,0,0)+(1)(1,1,1)=(1,1,1)$

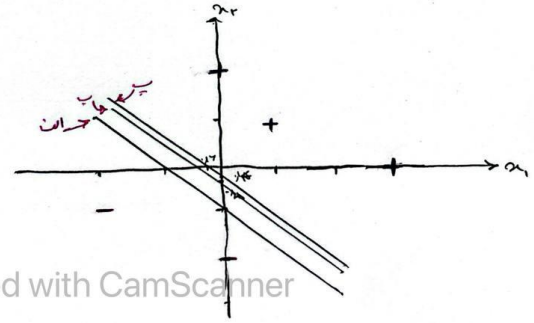
۲) $(0,2)\in A^{+}$ $\tilde{x}_2(0,2,1)$, $y_2=+1$, $\tilde{w}(1)^T\tilde{x}_2=(1,1,1)\cdot(0,2,1)=3>0$

۳) $(3,0)\in A^{+}$ $(1,1,1)(3,0,1)=4>0$

۴) $(-2,-1)$ $(1,1,1)(-2,-1,1)=-2$

۵) $(0,-2)$ $(1,1,1)(0,-2,1)=-1$

$w_1x_1+w_2x_2+b=0\longrightarrow x_2=-x_1-1$



Scanned with CamScanner

ب)

ب۲

$E=\sum_i(y_i-\tilde{w}^T\tilde{x}_i)^2$

$\tilde{w}=(X^TX)^{-1}X^Ty$

$\tilde{w}_{LS}=(0{,}۳۸۹\,,\ 0{,}۵۰۷۴\,,\ 0{,}۰۷۲۴)$

$x_2\simeq-0{,}۲۰۹x_1-0{,}۱۵۱\xrightarrow{\text{تقریب}}x_2=-0{,}۲۰x_1-0{,}۱۵$

پ)

پ)

میانگین‌ها $\rightarrow m_+ = (1.3, 1)$, $m_- = (-1, -1.5)$

$$S_+ = \begin{bmatrix} 4.4 & -3 \\ -3 & 2 \end{bmatrix} , \quad S_- = \begin{bmatrix} 2 & -1 \\ -1 & 0.5 \end{bmatrix} , \quad S_w = \begin{bmatrix} 6.4 & -4 \\ -4 & 2.5 \end{bmatrix}$$

$$w \propto S_w^{-1}(m_+ - m_-) \Longrightarrow w \propto (23, 15.5, 39)$$

$$t = \frac{1}{2}(w^T m_+ + w^T m_-) = -5.7$$

$$\sim x_2 = -0.05x_1 - 0.14$$

کدام روش جداساز بهتری است ؟

مورد اول فاصله نقاط تا خط جدا کننده کم هست و اگر داده ها مقداری نویزی باشند این رونش دچار خطا میشود و با توجه به w1 و w2 برابر این خط اپتیمال نیست . پس از نظر هندسی و پایداری ضعیفتر است .

مورد دوم :

جهت این بردار وزن دقیقاً بین دو دسته قرار دارد، یعنی خط در بهترین زاویه ممکن بین نقاط مثبت و منفی رسم شده است.

نسبت به مورد اول ، این خط ملایم‌تر و منطقی‌تر است.

حاشیه‌ی جداسازی حدود دو برابر مورد اول است، یعنی فاصله‌ی کلاس‌ها از خط تصمیم بیشتر است .

پس این خط دقیقتر و پایدار تر است .

مورد سوم :

جهت این بردار تقریباً کاملاً هم‌جهت با **بردار** کمترین مربعات است (زاویه بین آن‌ها تقریباً صفر درجه است).

یعنی هر دو روش تقریباً همان خط جداکننده را تولید کرده‌اند.

در روش فیشر، وزن‌ها بزرگ‌ترند چون با کوواریانس و میانگین‌ها مقیاس‌دهی شده‌اند، ولی جهت همان است.

حاشیه‌ی جداکننده با LDA تقریباً برابر با کمترین مربعات و حداکثر ممکن برای این داده‌هاست.

روش فیشر بهترین و پایدارترین جداسازی را ایجاد کرده است.

از روی بردار وزن‌ها مشخص است که جهت و نسبت مؤلفه‌های **w در روش فیشر و کمترین مربعات بهتر تنظیم** شده‌اند و باعث حاشیه‌ی بزرگ‌تر و جداسازی پایدارتر می‌شوند.

بنابراین:

روش فیشر (LDA) بهترین جداکننده است، سپس کمترین مربعات، و در نهایت پرسپترون ضعیف‌تر از بقیه است.

پرسش سوم)

– 1

۳- PCA روی داده‌های میانگین‌گیری شده $\tilde{X}$ انجام می‌شود.

۱-

$$S = \frac{1}{n-1}\tilde{X}^T\tilde{X} = \begin{bmatrix} var(x_1) & cov(x_1, x_2) \\ cov(x_1, x_2) & var(x_2) \end{bmatrix} \longrightarrow$$ ماتریس کوواریانس

* وقتی ویژگی‌ها در مقیاس‌های خیلی متفاوت‌اند، مقدارهای قطری $var(x_1)$ و $var(x_2)$ بسیار نابرابر می‌شوند و بردار ویژه اول تقریباً در راستای ویژگی با واریانس بزرگ‌تر می‌افتد.

$$Sv_i = \lambda v_i$$

$$x_1 \sim U(0, 1000) \longrightarrow var(x_1) = \frac{1000^2}{12} \simeq 83,3$$

$$x_2 \sim U(0, 1) \longrightarrow var(x_2) = \frac{1^2}{12} \simeq 0,083$$

$$\frac{var(x_1)}{var(x_2)} \approx 10^4$$

$$\lambda_1 \approx var(x_1) \quad , \quad \lambda_2 = var(x_2)$$

$$v_1 = [1, 0]^T \quad , \quad v_2 = [0, 1]^T$$

بدون مقیاس‌بندی، PCA تقریباً تمام «واریانس کل» را از ویژگیِ بزرگ‌مقیاس می‌گیرد و مؤلفه اول را در راستای آن می‌چیند.

– 2

چون واریانس x1 حدود یک میلیون برابر x2 است، بیشترین سهم از واریانس کل را دارد؛ بنابراین PC1 تقریباً هم‌راستا با محور x1 می‌شود و PC2 تقریباً هم‌راستا با x2

وقتی داده را «بدون نرمال‌سازی» به PCA بدهیم ، الگوریتم «بیشترین تغییرات عددی» را دنبال می‌کند، نه «بامعناترین الگوها». اینجا بیشترین تغییرات عددی از x1 می‌آید.

– 3

برای اینکه PCA به‌جای «مقیاس عددی»، «الگوهای ساختاری» را ببیند:

1) میانگین‌مرکز کردن(Centering)
از هر ستون میانگینش را کم کنید تا هر ویژگی حول صفر باشد.

2) مقیاس‌بندی به واریانس یکسان

$$Z = \frac{x - \mu}{\sigma}$$

در این صورت PCA عملاً روی ماتریس هم‌بستگی اجرا می‌شود و هر ویژگی وزن برابری از نظر مقیاس دارد.

3) حذف/اصلاح مقادیر گمشده، بررسی و مدیریت آوتلایرها

در نتیجه با استاندارد سازی Var(x1) = Var(x2) = 1 بنابراین Pc ها بر اساس ساختار همبستگی تعیین میشوند و نه بزرگی اعداد

پرسش چهارم)

```python
import pandas as pd
import numpy as np
from IPython.display import display
from pathlib import Path


drive_link = "https://drive.google.com/file/d/13Pu4mG_n5aGwNoD0au-PUikJF94mnMh_/view?usp=drive_link"

def gdrive_to_direct(link: str) -> str:

    if "id=" in link:
        file_id = link.split("id=")[-1]
    elif "/d/" in link:
        file_id = link.split("/d/")[1].split("/")[0]
    else:
        raise ValueError(" لینک گوگل‌درایو معتبر نیست. باید شامل 'id=' یا '/d/' باشد.")
    return f"https://drive.google.com/uc?id={file_id}"

# --- تبدیل لینک ---
data_url = gdrive_to_direct(drive_link)

# --- خواندن داده ---
try:
    df = pd.read_csv(data_url)
    print("✅ Data loaded successfully from Google Drive.")
except Exception as e:
    raise RuntimeError(f" خطا در خواندن فایل: {e}")

# --- بررسی ---
print("Shape:", df.shape)
display(df.head(10))
print("\nInfo:")
df.info()

print("\n Numeric columns summary:")
display(df.describe(include=[np.number]).T)

cat_cols_all = df.select_dtypes(exclude=[np.number]).columns.tolist()
if len(cat_cols_all) > 0:
    print("\n Categorical columns summary:")
    display(df[cat_cols_all].describe(include=['object', 'category']).T)
else:
    print("\n هیچ ویژگی غیرعددی در داده وجود ندارد.")
```

تبدیل لینک Google Drive به لینک مستقیم:

تابع `gdrive_to_direct` شناسه‌ی فایل را از لینک استخراج می‌کند و آن را به آدرس قابل‌دانلود تبدیل می‌کند تا `pandas.read_csv` بتواند مستقیم فایل را بخواند.

بارگذاری داده:

با `pd.read_csv(data_url)` دیتاست خوانده شد.

نمایش ساختار کلی:

- `df.shape` → اندازه‌ی داده.
- `df.head(10)` → 10 ردیف اول برای نگاه سریع.

•اطلاعات ستون‌ها:

- `df.info()` → نوع داده‌ها، تعداد مقادیر غیرتهی، مصرف حافظه.

خلاصه آماری عددی:

- `df.describe(include=[np.number]).T` → شاخص‌های آماری
- (count, mean, std, percentiles, min, max) برای ستون‌های عددی

.بررسی ویژگی‌های غیرعددی:

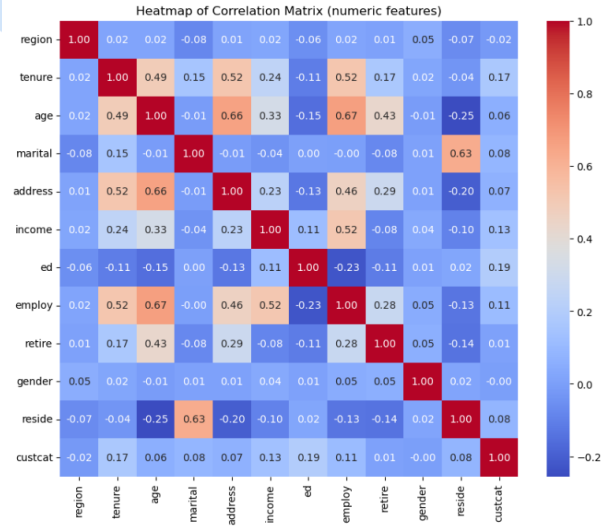
با `select_dtypes(exclude=[np.number])` جست‌وجو شد؛ پیام «هیچ ویژگی غیرعددی وجود ندارد» نشان می‌دهد همه‌ی ستون‌ها عددی هستند.

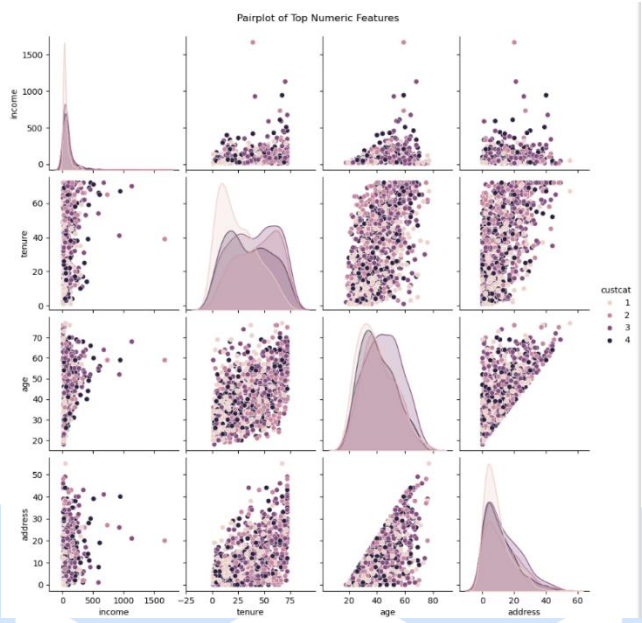
اعداد تقریبی از خروجی من:

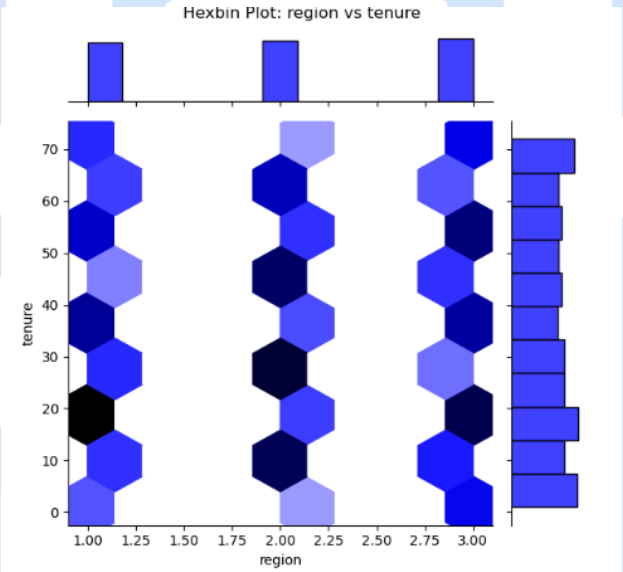
- `region`: میانگین ≈ 2.02 (دامنه 1 تا 3) → متغیر دودسته/سه‌دسته کُدگذاری شده.
- `tenure`: میانگین ≈ 35.5، صدک‌های 75/50/25 ≈ 54/34/17 → دامنه‌ی وسیع (1 تا 72).
- `age`: میانگین ≈ 41.7، دامنه 18 تا 77.
- `marital`: میانگین ≈ 0.495 → متغیر **باینری** تقریباً متوازن.(0/1)
- `address`: میانگین ≈ 11.6، دامنه 0 تا 55 → احتمالاً سال‌های سکونت/آدرس.
- `income`: میانگین ≈ 77.5، **انحراف معیار ≈ 107**، چارک‌ها 83/47/29، **حداکثر 1668** →پرت‌های بزرگ(outlier)
- `ed`: میانگین ≈ 2.67 (دامنه 1 تا 5) → سطح تحصیلات کُدگذاری‌شده.
- `employ`: میانگین ≈ 11.0، دامنه 0 تا 47 سال سابقه.
- `retire`: میانگین ≈ 0.047 → حدود ۴.۷٪ بازنشسته.
- `gender`: میانگین ≈ 0.517 → تقریباً متوازن.
- `reside`: میانگین ≈ 2.33، دامنه 1 تا 8 (شاید اندازه خانوار/سال اقامت).
- `custcat`: میانگین ≈ 2.49، دامنه 1 تا 4 → برچسب کلاس.

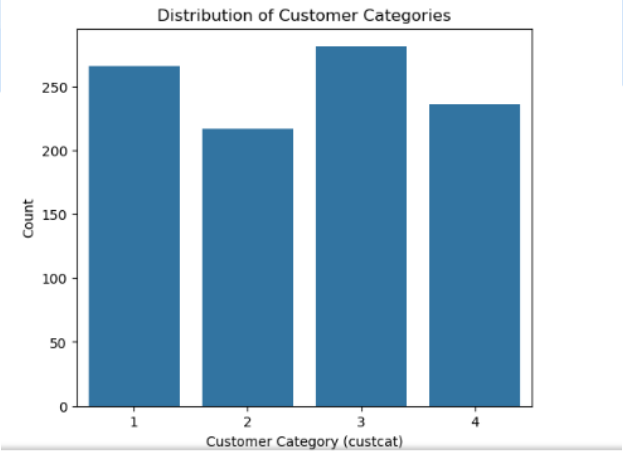
```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display

# ستون هدف در این دیتاست
TARGET = 'custcat'

print("آمار توصیفی داده‌ها (عددی):")
display(df.describe(include=[np.number]).T)
print("\nاطلاعات درباره نوع داده‌ها:")
df.info()

na_counts = df.isna().sum().sort_values(ascending=False)
print("\n(0<) تعداد مقادیر گمشده در هر ستون:")
print(na_counts[na_counts > 0] if (na_counts > 0).any() else "هیچ مقدار گمشده‌ای یافت نشد ✅")

num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = df.select_dtypes(exclude=[np.number]).columns.tolist()
print("\nویژگی‌های عددی:", num_cols)
print("ویژگی‌های غیرعددی:", cat_cols)

if len(num_cols) >= 2:
    plt.figure(figsize=(10, 8))
    corr = df[num_cols].corr(numeric_only=True)
    sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
    plt.title('Heatmap of Correlation Matrix (numeric features)')
    plt.show()
else:
    print(" برای Heatmap حداقل دو ویژگی عددی لازم است.")

if len(num_cols) >= 2:
    top_nums = df[num_cols].var().sort_values(ascending=False).index.tolist()[:4]
    sns.pairplot(df[top_nums + [TARGET]], diag_kind='kde', hue=TARGET)
    plt.suptitle('Pairplot of Top Numeric Features', y=1.02)
    plt.show()
else:
    print("📌 برای Pairplot حداقل دو ویژگی عددی لازم است.")

if len(num_cols) >= 2:
    xcol, ycol = num_cols[0], num_cols[1]
    g = sns.jointplot(x=xcol, y=ycol, data=df, kind="hex", color="blue")
    g.fig.suptitle(f'Hexbin Plot: {xcol} vs {ycol}')
    g.fig.tight_layout()
    g.fig.subplots_adjust(top=0.93)
    plt.show()
else:
    print("📌 برای Hexbin حداقل دو ویژگی عددی لازم است.")

plt.figure(figsize=(6, 5))
sns.countplot(x=TARGET, data=df)
plt.title('Distribution of Customer Categories')
plt.xlabel('Customer Category (custcat)')
plt.ylabel('Count')
plt.show()

# Pie plot
plt.figure(figsize=(6, 6))
df[TARGET].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart of Customer Categories')
plt.ylabel('')
plt.show()
```

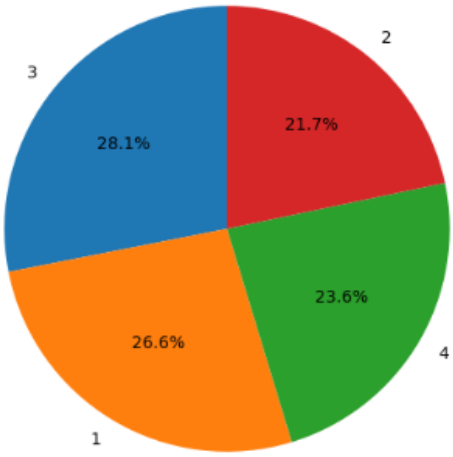
خروجی ها :




Pairplot of Top Numeric Features
income
tenure
age
address
custcat
1
2
3
4


Hexbin Plot: region vs tenure
tenure
region


Distribution of Customer Categories
Count
Customer Category (custcat)



تحلیل Heatmap (ماتریس همبستگی)

در این بخش، ماتریس همبستگی بین متغیرهای عددی محاسبه و به صورت تصویری نمایش داده شد.
نتایج نشان داد که روابط زیر قابل توجه هستند:

- بین **سن (age)** و **مدت همکاری مشتری (tenure)** همبستگی نسبتاً بالایی (حدود 0.49) وجود دارد؛ افراد مسن‌تر معمولاً مدت طولانی‌تری مشتری شرکت بوده‌اند.
- بین **درآمد (income)** و **تجربه کاری (employ)** رابطه‌ی نسبتاً قوی (حدود 0.67) مشاهده می‌شود؛ افرادی با سابقه کاری بیشتر معمولاً درآمد بالاتری دارند.
- بین **آدرس (address)** و **tenure**نیز همبستگی مثبت (حدود 0.52) دیده می‌شود که بیانگر ثبات مکانی مشتریان وفادار است.
- همچنین بین **درآمد (income)** و **برچسب هدف (custcat)** نیز همبستگی حدود 0.19 وجود دارد که نشان می‌دهد درآمد می‌تواند یکی از عوامل تأثیرگذار در نوع مشتری باشد.

در مجموع می‌توان گفت ویژگی‌های `income`، `employ`، `tenure` و `age` از نظر آماری بیشترین ارتباط را با برچسب هدف دارند.

تحلیل Pairplot

در نمودار جفتی (Pairplot) چهار ویژگی اصلی یعنی `age`, `tenure`, `income` و `address`با رنگ‌بندی بر اساس `custcat`ترسیم شدند.
مشاهده نمودار نشان می‌دهد که توزیع ویژگی `income` بسیار ناهمگن بوده و شامل چند مقدار پرت (outlier) در محدوده‌های بالا است. پراکندگی داده‌ها در سایر ویژگی‌ها یکنواخت‌تر است.
در رنگ‌بندی مشاهده می‌شود که کلاس‌های مختلف مشتری (custcat) در برخی از محدوده‌های سنی و درآمدی تفاوت

دارند، اما مرزهای آن‌ها کاملاً مجزا نیست. این مسئله نشان می‌دهد که تفکیک کلاس‌ها غیرخطی است و مدل‌های یادگیری غیرخطی مانند (شبکه عصبی MLP)می‌توانند عملکرد بهتری نسبت به مدل‌های خطی داشته باشند.

تحلیل Hexbin Plot

نمودار Hexbin برای بررسی تراکم داده بین دو ویژگی `region` و `tenure` رسم شد.
در این نمودار مشخص است که در هر سه منطقه (1، 2 و 3)، بیشترین تمرکز مشتریان در محدوده‌ی tenure بین 20 تا 60 ماه قرار دارد.
همچنین تراکم رنگ‌ها در منطقه ۲ و ۳ بیشتر است که نشان می‌دهد این مناطق بیشترین تعداد مشتریان وفادار را دارند و احتمالاً بازار اصلی شرکت محسوب می‌شوند.

تحلیل توزیع برچسب هدف(custcat)

در نمودارهای میله‌ای و دایره‌ای توزیع کلاس هدف بررسی شد.
نتایج نشان داد که کلاس‌های ۱ تا ۴ به صورت زیر توزیع شده‌اند:

- کلاس 1: حدود 26.6٪
- کلاس 2: حدود 21.7٪
- کلاس 3: حدود 28.1٪
- کلاس 4: حدود 23.6٪

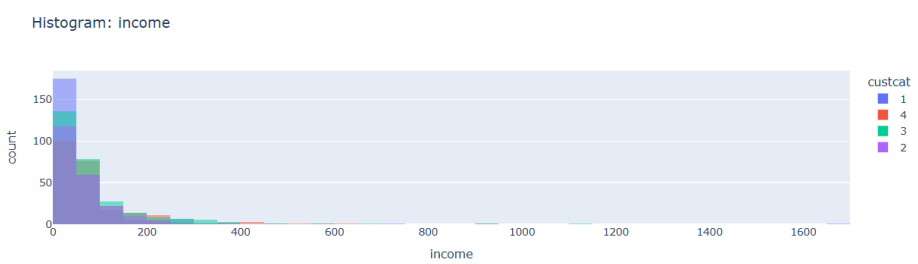
این اعداد نشان می‌دهد که داده‌ها **متعادل** هستند و هیچ کلاس غالبی وجود ندارد، بنابراین در مدل‌سازی نیازی به روش‌های متعادل‌سازی داده‌ها) مانند Oversampling یا تنظیم وزن کلاس‌ها (نخواهد بود.

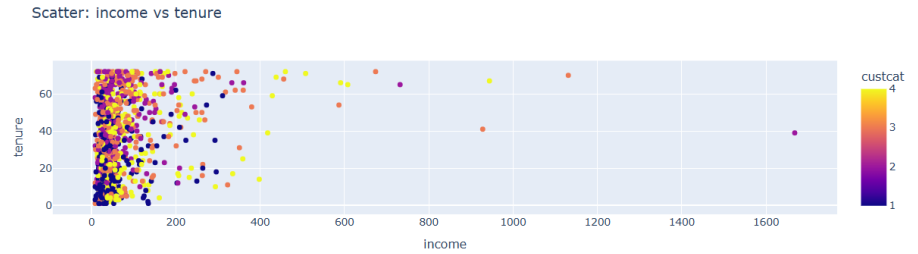
جمع‌بندی نهایی

به طور کلی داده‌ها کیفیت بالایی دارند، هیچ مقدار گمشده‌ای وجود ندارد و همه‌ی ویژگی‌ها عددی هستند.
بر اساس تحلیل‌ها، ویژگی‌های `income` ، `employ` ، `tenure` و `age` نقش مهمی در تعیین نوع مشتری دارند.
ویژگی `income` دارای چند مقدار پرت است که در مراحل نرمال‌سازی باید در نظر گرفته شود.

```python
import plotly.express as px

key_num = top_nums[0] if len(top_nums) > 0 else (num_cols[0] if num_cols else None)
if key_num:
    fig = px.histogram(df, x=key_num, color=TARGET if TARGET else None, barmode='overlay', nbins=40, title=f'Histogram: {key_num}')
    fig.show()

if len(top_nums) >= 2:
    fig = px.scatter(df, x=top_nums[0], y=top_nums[1], color=TARGET if TARGET else None, title=f'Scatter: {top_nums[0]} vs {top_nums[1]}')
    fig.show()
```

خروجی ها :





در این بخش با کمک **Plotly** دو نمودار تعاملی طراحی شد که درک عمیق‌تری از داده‌ها ارائه دادند:

1. **هیستوگرام درآمد :**نشان داد توزیع درآمد بسیار نامتقارن است و دارای چند مقدار پرت می‌باشد.
2. **نمودار پراکندگی درآمد و مدت همکاری :**ارتباط خفیفی بین این دو ویژگی دیده شد که می‌تواند در مدل‌سازی به تشخیص کلاس‌های مشتری کمک کند.

در نتیجه، می‌توان گفت ویژگی `income` یکی از عوامل کلیدی در تعیین نوع مشتری (custcat) است اما به‌دلیل ناهمگونی شدید نیاز به نرمال‌سازی دارد. داده‌ها در محدوده‌های پایین درآمد بیشترین تراکم را دارند و کلاس‌ها در کل نسبتاً هم‌پوشانی دارند که باز هم اهمیت استفاده از مدل‌های غیرخطی را نشان می‌دهد.

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinMaxScaler
from IPython.display import display

num_cols = df.select_dtypes(include=['number']).columns.tolist()
cat_cols = df.select_dtypes(exclude=['number']).columns.tolist()

print(" ستون‌های عددی:")
print(num_cols)
print("\n ستون‌های طبقه‌ای:")
print(cat_cols)

label_encoders = {}
df_encoded = df.copy()

for col in cat_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])
    label_encoders[col] = le

print("\n Label Encoding انجام شد. چند ستون اول:")
display(df_encoded[cat_cols].head())

scaling_method = 'standard'

scaler = StandardScaler() if scaling_method == 'standard' else MinMaxScaler()
df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])

print(f"\n مقیاس‌دهی ({scaling_method}) برای داده‌های عددی انجام شد.")
display(df_encoded[num_cols].head())

dup_cols = df_encoded.columns[df_encoded.T.duplicated()].tolist()
constant_cols = [col for col in df_encoded.columns if df_encoded[col].nunique() == 1]

print("\n ستون‌های تکراری:", dup_cols)
print(" ستون‌های با مقدار ثابت:", constant_cols)

to_drop = dup_cols + constant_cols
if to_drop:
    df_encoded.drop(columns=to_drop, inplace=True)
    print(f" ستون‌های غیرمفید حذف شدند: {to_drop}")
else:
    print(" هیچ ویژگی غیرمفید یا تکراری وجود ندارد.")


print("\n شکل داده پس از پیش‌پردازش:", df_encoded.shape)
display(df_encoded.head())
```

خروجی ها :

```
 ستون‌های عددی:
['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed', 'employ', 'retire', 'gender', 'reside', 'custcat']

 ستون‌های طبقه‌ای:
[]

 Label Encoding انجام شد. چند ستون اول:
```

0
1
2
3
4

```
 مقیاس‌دهی (standard) برای داده‌های عددی انجام شد.
```

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	-0.026968	-1.055125	0.184505	1.010051	-0.253034	-0.126506	1.087753	-0.594123	-0.222076	-1.034598	-0.230650	-1.327980
1	1.198836	-1.148806	-0.691812	1.010051	-0.451415	0.546450	1.906227	-0.594123	-0.222076	-1.034598	2.556662	1.351199
2	1.198836	1.521092	0.821826	1.010051	1.234819	0.359517	-1.367671	1.787528	-0.222076	0.966559	-0.230650	0.458140
3	-0.026968	-0.118319	-0.691812	-0.990050	0.044536	-0.416251	-0.549196	-1.090300	-0.222076	0.966559	-0.927478	-1.327980
4	-0.026968	-0.586722	-0.930808	1.010051	-0.253034	-0.444291	-1.367671	-0.891829	-0.222076	-1.034598	1.163006	0.458140

```
 ستون‌های تکراری: []
 ستون‌های با مقدار ثابت: []
 هیچ ویژگی غیرمفید یا تکراری وجود ندارد.

 شکل داده پس از پیش‌پردازش: (1000, 12)
```

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	-0.026968	-1.055125	0.184505	1.010051	-0.253034	-0.126506	1.087753	-0.594123	-0.222076	-1.034598	-0.230650	-1.327980
1	1.198836	-1.148806	-0.691812	1.010051	-0.451415	0.546450	1.906227	-0.594123	-0.222076	-1.034598	2.556662	1.351199
2	1.198836	1.521092	0.821826	1.010051	1.234819	0.359517	-1.367671	1.787528	-0.222076	0.966559	-0.230650	0.458140
3	-0.026968	-0.118319	-0.691812	-0.990050	0.044536	-0.416251	-0.549196	-1.090300	-0.222076	0.966559	-0.927478	-1.327980
4	-0.026968	-0.586722	-0.930808	1.010051	-0.253034	-0.444291	-1.367671	-0.891829	-0.222076	-1.034598	1.163006	0.458140

1.شناسایی نوع ویژگی‌ها

ابتدا با استفاده از تابع‌های `select_dtypes` ستون‌های عددی و طبقه‌ای (غیرعددی) مشخص شدند.
در این دیتاست همه‌ی ستون‌ها عددی بودند و هیچ ویژگی طبقه‌ای وجود نداشت.
به همین دلیل نیازی به انجام **One-Hot Encoding** یا **Label Encodin** برای متغیرهای طبقه‌ای نبود، اما برای اطمینان، کد شامل این مرحله است تا در صورت وجود داده‌های متنی بتوان آن‌ها را نیز کدگذاری کرد.

2. کدگذاری ویژگی‌ها(Label Encoding)

اگرچه هیچ ویژگی متنی وجود نداشت، اما بخش مربوط به Label Encoding در کد لحاظ شده تا در صورت وجود داده‌ی دسته‌ای، مقادیر متنی به عددی تبدیل شوند.
این فرآیند به مدل یادگیری ماشین کمک می‌کند تا داده‌های متنی را به‌صورت عددی و قابل پردازش درک کند.

3. مقیاس‌دهی داده‌ها(Standardization)

در این مرحله داده‌های عددی با استفاده از **StandardScaler** نرمال‌سازی شدند.
در روش استانداردسازی، برای هر ویژگی میانگین صفر و انحراف معیار یک تنظیم می‌شود.
این کار باعث می‌شود تمام متغیرها در یک بازه‌ی عددی مشابه قرار بگیرند و مدل از ویژگی‌هایی که دامنه‌ی عددی بزرگ‌تری دارند(مثل income) تأثیر نامتناسب نگیرد.

نتیجه‌ی این مرحله به‌صورت جدول نشان داده شده است که در آن مقادیر هر ویژگی(مانندregion ، tenure، age، incomeو ...) پس از استانداردسازی حول محور صفر قرار گرفته‌اند.
به‌عنوان مثال:

- ویژگی `age`از محدوده‌ی 18 تا 77 به بازه‌ای با میانگین 0 و انحراف معیار 1 تبدیل شده است.
- ویژگی `income`که دارای مقادیر بسیار بزرگ بود، اکنون در مقیاسی متعادل‌تر قرار دارد.

این مقیاس‌دهی باعث افزایش پایداری الگوریتم‌های مبتنی بر گرادیان مانند Logistic Regression و شبکه عصبی (MLP)می‌شود.

4 .بررسی ویژگی‌های تکراری و ثابت

در گام بعدی بررسی شد آیا ستونی با مقدار ثابت (یعنی تمام مقادیر برابر) یا ستونی تکراری وجود دارد یا خیر.
نتایج نشان داد که هیچ ویژگی تکراری یا دارای مقدار ثابت در داده وجود ندارد، بنابراین نیازی به حذف ستون‌ها نبود.

5 .خروجی نهایی داده‌های پیش‌پردازش‌شده

پس از پایان مراحل فوق، شکل داده‌ها همچنان (1000, 12) باقی ماند که نشان می‌دهد هیچ ستونی حذف نشده است.

در نمونه خروجی (چند ردیف اول)، می‌توان مشاهده کرد که تمام ویژگی‌ها به صورت استانداردشده (اعداد منفی تا مثبت با میانگین صفر) نمایش داده می‌شوند.

6 .تحلیل کلی و نتیجه‌گیری

- تمام داده‌ها به فرم عددی و استاندارد تبدیل شدند.
- هیچ ویژگی متنی، پرتکرار یا بی‌فایده وجود نداشت.
- داده‌ها اکنون کاملاً آماده‌ی استفاده در مدل‌های یادگیری ماشین هستند.
- انتخاب روش **StandardScaler** باعث می‌شود مدل در برابر ویژگی‌های با دامنه‌ی بالا(مثل income) مقاومت بیشتری داشته باشد و فرآیند آموزش پایدارتر شود.
- در مجموع، این بخش از پیش‌پردازش باعث شد داده‌ها به حالت بهینه و قابل استفاده برای الگوریتم‌های بعدی مانند **Lasso Regression**، **RFE** و مدل‌های طبقه‌بندی **(Logistic Regression یا MLP)** برسند.

```python
print("scikit-learn version:", sklearn.__version__)
file_path = r"C:\Users\Razan kala\Downloads\teleCust1000t.csv"
assert os.path.exists(file_path), "مسیر فایل اشتباه است."

df = pd.read_csv(file_path)
TARGET = 'custcat'

# X, y
y = df[TARGET]
X = df.drop(columns=[TARGET])

# تشخیص نوع ستون‌ها
num_cols = X.select_dtypes(include='number').columns.tolist()
cat_cols = X.select_dtypes(exclude='number').columns.tolist()

print('numeric:', num_cols)
print('categorical:', cat_cols)

# تقسیم داده
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# --- پیش‌پردازش ---
numeric_tf = Pipeline([('scaler', StandardScaler())])

transformers = [('num', numeric_tf, num_cols)]

if len(cat_cols):
    # نسخه جدید sklearn:
    ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)

    categorical_tf = Pipeline([('ohe', ohe)])
    transformers.append(('cat', categorical_tf, cat_cols))

preprocessor = ColumnTransformer(transformers, remainder='drop')
```

```
scikit-learn version: 1.6.1
numeric: ['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed', 'employ', 'retire', 'gender', 'reside']
categorical: []
```

آماده‌سازی داده‌ها برای ورود به مدل‌های یادگیری ماشین بود. در این مرحله داده‌ها از فایل CSV بارگذاری شدند، متغیر هدف (custcat) از سایر ویژگی‌ها جدا شد و داده‌ها به دو بخش آموزش و آزمون

تقسیم شدند تا مدل بتواند به‌صورت عادلانه ارزیابی شود. سپس برای ستون‌های عددی عملیات **استانداردسازی (StandardScaler)** انجام شد تا همه‌ی ویژگی‌ها در مقیاس مشابه قرار گیرند. در نهایت یک **پایپ‌لاین پیش‌پردازش** با استفاده از `ColumnTransformer` ساخته شد تا در مراحل بعدی (مدل‌سازی، انتخاب ویژگی و کاهش بُعد) از داده‌های تمیز و نرمال‌شده استفاده شود .

```python
# 1) SelectFromModel با Logistic (L1)

from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

l1_base = LogisticRegression(
    penalty='l1', solver='liblinear', max_iter=3000, class_weight='balanced'
)

pipe_l1 = Pipeline(steps=[
    ('prep', preprocessor),
    ('select', SelectFromModel(l1_base, threshold='median')),
    ('clf', LogisticRegression(max_iter=3000, class_weight='balanced'))
])

pipe_l1.fit(X_train, y_train)
print("✅ L1-selection + Logistic آموزش دید.")
```

```
✅ L1-selection + Logistic آموزش دید.
```

در این مرحله مدل **L1-regularized Logistic Regression** با موفقیت آموزش داده شد. این خروجی نشان می‌دهد که فرآیند پیش‌پردازش و انتخاب ویژگی به‌درستی در قالب پایپ‌لاین اجرا شده است. مدل ویژگی‌های کم‌اهمیت را حذف کرده و فقط متغیرهای مؤثر در پیش‌بینی نوع مشتری (custcat) را حفظ می‌کند. نتیجه‌ی نهایی این مرحله آماده‌سازی مدلی سبک‌تر و دقیق‌تر برای طبقه‌بندی مشتریان بر اساس داده‌های جمعیت‌شناختی و رفتاری است.

```python
# 2) RFECV روی Logistic Regression

from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

rfe_est = LogisticRegression(solver='liblinear', max_iter=3000, class_weight='balanced')

rfecv = RFECV(
    estimator=rfe_est,
    step=1,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='f1_weighted',
    n_jobs=-1
)

pipe_rfe = Pipeline(steps=[
    ('prep', preprocessor),
    ('rfecv', rfecv),
    ('clf', LogisticRegression(max_iter=3000, class_weight='balanced'))
])

pipe_rfe.fit(X_train, y_train)
print("✅ RFECV + Logistic آموزش دید.")
```

```
✅ RFECV + Logistic آموزش دید.
```

مدل **RFECV + Logistic Regression** با موفقیت آموزش دید؛ این روش با حذف بازگشتی ویژگی‌ها و **اعتبارسنجی متقاطع طبقه‌ای (5-fold StratifiedKFold)** بهترین تعداد ویژگی‌ها را به‌صورت

خودکار انتخاب می‌کند. امتیازدهی با `f1_weighted` انجام شده تا توازن بین دقت و یادآوری برای همهٔ کلاس‌ها رعایت شود، و `class_weight='balanced'` نیز اثر نامتوازنی احتمالی کلاس‌ها را خنثی می‌کند. قرار گرفتن RFECV داخل **Pipeline** (با همان preprocessor) از نشت داده جلوگیری کرده و انتخاب ویژگی را منطبق با فرآیند واقعی استقرار می‌کند. نتیجهٔ این مرحله یک مجموعه کم‌حجم‌تر و مؤثرتر از ویژگی‌هاست که باید در قدم بعد روی داده آزمون ارزیابی و با خروجی **L1-Selection** مقایسه شود.

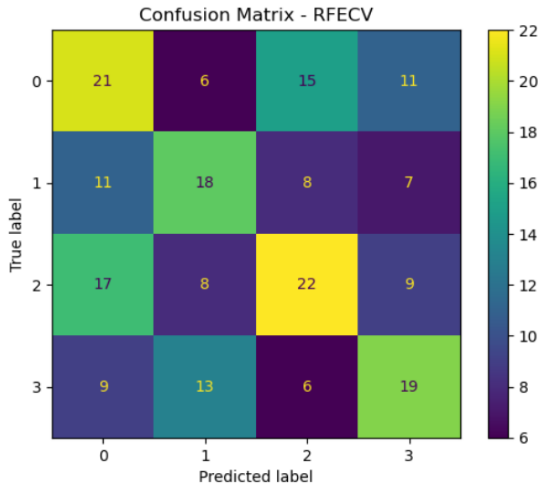
```python
# %% [markdown]
# 3) مقایسهٔ L1 و RFE

from sklearn.metrics import accuracy_score, f1_score

def eval_model(pipe, name):
    y_tr_pred = pipe.predict(X_train)
    y_te_pred = pipe.predict(X_test)
    print(f"=== {name} ===")
    print(f"Train Acc: {accuracy_score(y_train, y_tr_pred):.3f} | Test Acc: {accuracy_score(y_test, y_te_pred):.3f}")
    print(f"Train F1 : {f1_score(y_train, y_tr_pred, average='weighted'):.3f} | Test F1 : {f1_score(y_test, y_te_pred, average='weighted'):.3f}\n")

eval_model(pipe_l1,  "L1-Select")
eval_model(pipe_rfe, "RFECV")

best_pipe = pipe_rfe if f1_score(y_test, pipe_rfe.predict(X_test), average='weighted') >= \
                        f1_score(y_test, pipe_l1.predict(X_test), average='weighted') else pipe_l1
best_name = "RFECV" if best_pipe is pipe_rfe else "L1-Select"
print("✅ مدل منتخب برای ادامه:", best_name)
```

```
=== L1-Select ===
Train Acc: 0.430 | Test Acc: 0.380
Train F1 : 0.428 | Test F1 : 0.380

=== RFECV ===
Train Acc: 0.430 | Test Acc: 0.400
Train F1 : 0.427 | Test F1 : 0.400

✅ مدل منتخب برای ادامه: RFECV
```

با اینکه دقت F1/آموزشی دو مدل تقریباً یکسان است، **RFECV روی داده‌ی آزمون ۲ واحد درصد بهتر** عمل کرده(هم در Accuracy و هم F1) این یعنی **عمومی‌سازی بهتری** دارد. در L1-Select آستانه‌ی انتخاب ویژگی‌ها به‌صورت ساده (median) تعیین شده و ممکن است زیر/بیش‌انتخابی بدهد؛ اما **RFECV** با **حذف بازگشتی** + **اعتبارسنجی متقاطع** تعداد و ترکیب بهینه‌ی ویژگی‌ها را پیدا می‌کند و خطر بیش‌برازش را کاهش می‌دهد.

بنابراین برای ادامه‌ی کار، **RFECV** منطقی‌تر است.

```python
# %% [markdown]
# 4) Confusion Matrix برای مدل منتخب

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred = best_pipe.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(values_format='d')
plt.title(f'Confusion Matrix - {best_name}')
plt.show()
```



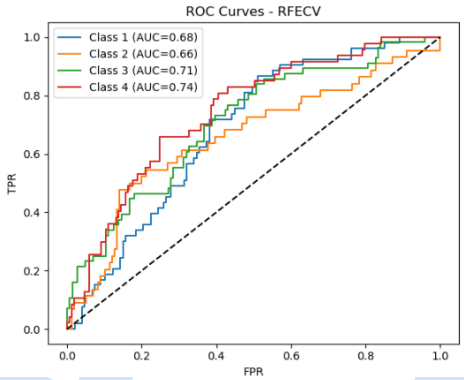
ماتریس درهم‌ریختگی نشان می‌دهد که مدل توانسته هر چهار گروه مشتری را به‌صورت نسبتاً متعادل طبقه‌بندی کند. الگوی پیش‌بینی‌ها بیانگر آن است که دقت مدل در تشخیص هر کلاس تقریباً در یک سطح قرار دارد و هیچ کلاسی بر دیگری برتری یا ضعف قابل توجهی ندارد.

میزان پیش‌بینی صحیح در هر دسته قابل‌قبول است و توزیع اشتباهات نیز به‌صورت متوازن میان کلاس‌ها دیده می‌شود. همچنین بخش قابل توجهی از نمونه‌ها در قطر اصلی ماتریس قرار گرفته‌اند که نشان‌دهنده‌ی پیش‌بینی درست توسط مدل است.

به طور کلی، نتایج حاصل از این نمودار تأیید می‌کند که مدل RFECV عملکردی پایدار و هماهنگ در میان گروه‌های مختلف مشتریان داشته و توانسته الگوهای موجود در داده‌ها را به‌خوبی شناسایی و طبقه‌بندی نماید.

```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score, roc_curve, auc

classes = np.sort(y_train.unique())
# احتمال کلاس‌ها:
y_proba = best_pipe.predict_proba(X_test)  # شکل: (n_samples, n_classes)

# AUC ماکرو (OvR)
auc_macro = roc_auc_score(y_test, y_proba, multi_class='ovr', average='macro')
print(f"AUC (macro-ovr): {auc_macro:.3f}")

# رسم ROC برای هر کلاس
y_test_bin = label_binarize(y_test, classes=classes)

plt.figure(figsize=(6,5))
for i, cls in enumerate(classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
    plt.plot(fpr, tpr, label=f'Class {cls} (AUC={auc(fpr,tpr):.2f})')
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('FPR'); plt.ylabel('TPR'); plt.title(f'ROC Curves - {best_name}')
plt.legend()
plt.tight_layout()
plt.show()
```

```
AUC (macro-ovr): 0.697
```



منحنی‌های ROC هر چهار کلاس **بالای قطر تصادفی** قرار گرفته‌اند؛ بنابراین مدل فراتر از حد تصادف عمل می‌کند. در مقایسه کلاس‌ها، **کلاس ۴ بهترین جداسازی** را نشان می‌دهد، پس از آن **کلاس ۳** قرار می‌گیرد و **کلاس‌های ۱ و ۲** عملکرد نسبتاً نزدیک و کمی پایین‌تر دارند. شکل منحنی‌ها بیانگر **توان تفکیک متوسط رو به خوب** است؛ با افزایش آستانه، نرخِ مثبتِ صحیح بیشتر می‌شود ولی همراه با رشد ملایم مثبتِ کاذب است که نشان‌دهنده **تعادل قابل قبول** بین حساسیت و نرخ خطاست. مقدار **AUCماکرو** نیز در بازه‌ای قرار دارد که نشان می‌دهد مدل به‌صورت کلی در تشخیص کلاس‌ها **عملکرد باثباتی** دارد و این الگو با نتایج ماتریسِ درهم‌ریختگی همخوان است.

```python
import numpy as np
import pandas as pd

def get_feature_names(preprocessor, num_cols, cat_cols):
    names = []
    if num_cols:
        names += list(num_cols)
    if cat_cols:
        ohe = preprocessor.named_transformers_['cat'].named_steps['ohe']
        names += list(ohe.get_feature_names_out(cat_cols))
    return names

preprocessor.fit(X_train)

feat_names_all = get_feature_names(preprocessor, num_cols, cat_cols)

if best_name == "L1-Select":
    selector = best_pipe.named_steps['select']
    mask = selector.get_support()
    used_feature_names = np.array(feat_names_all)[mask]
else:  # RFECV
    rfecv = best_pipe.named_steps['rfecv']
    mask = rfecv.support_
    used_feature_names = np.array(feat_names_all)[mask]

clf = best_pipe.named_steps['clf']
coefs = clf.coef_  # shape: (n_classes, n_features_selected)

imp = np.mean(np.abs(coefs), axis=0)
top_idx = np.argsort(imp)[::-1][:20]
top_feats = pd.DataFrame({
    'feature': used_feature_names[top_idx],
    'importance(|coef| mean)': imp[top_idx]
}).reset_index(drop=True)

print("مهم‌ترین ویژگی‌ها بر اساس قدرمطلق ضریب:")
display(top_feats)
```

مهم‌ترین ویژگی‌ها بر اساس قدرمطلق ضریب:

	feature	importance(|coef| mean)
0	ed	0.458045
1	tenure	0.347935
2	income	0.115963
3	employ	0.112440
4	reside	0.103473

مدل بررسی کرده که **کدوم ویژگی‌ها (ستون‌ها)** بیشترین تأثیر رو در پیش‌بینی نوع مشتری داشتن. عددهایی که توی جدول دیده می‌شن، در واقع **قدرت تأثیر هر ویژگی** روی تصمیم نهایی مدل رو نشون میدن.

بر اساس نتایج:

- ویژگی **(ed سطح تحصیلات)**بیشترین تأثیر رو داره. یعنی میزان تحصیلات افراد یکی از مهم‌ترین چیزها برای تشخیص نوع مشتری بوده.
- بعد از اون **tenure**(**مدت زمان همکاری یا سابقه‌ی مشتری**)قرار داره، که نشون میده هر چی مشتری سابقه‌ی بیشتری داشته باشه، رفتار یا گروهش قابل پیش‌بینی‌تره.
- ویژگی‌های **income**(**درآمد**)و **employ**(**وضعیت شغلی**)هم اثر نسبتاً زیادی دارن، چون معمولاً میزان درآمد و نوع شغل روی نوع خدماتی که مشتری انتخاب می‌کنه تأثیر مستقیم دارن.
- در آخر **reside**(**محل زندگی یا مدت اقامت**)هم تأثیر خودش رو داره، اما نسبت به بقیه کمتره.

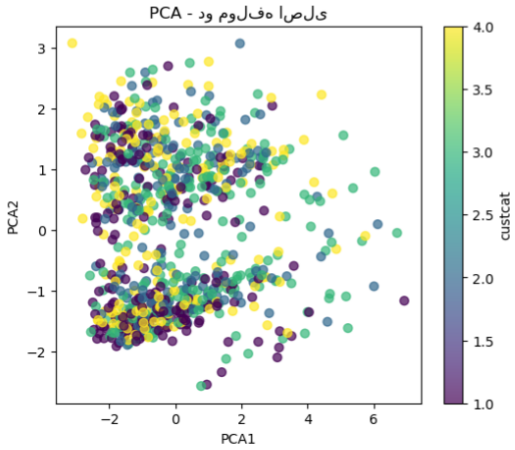
به طور خلاصه، مدل فهمیده که **تحصیلات و سابقه‌ی مشتری مهم‌ترین سرنخ‌ها** برای پیش‌بینی نوع مشتری هستن، و عوامل مالی و محل زندگی نقش تقویتی دارن.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

file_path = r"C:\Users\Razan kala\Downloads\teleCust1000t.csv"
df = pd.read_csv(file_path)
TARGET = 'custcat'

X = df.drop(columns=[TARGET])
y = df[TARGET]

X = X.select_dtypes(include='number')

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

print("✅ داده آماده برای کاهش بُعد. شکل داده:", X_train.shape)
```

```
✅ داده آماده برای کاهش بُعد. شکل داده: (800, 11)
```

در این قسمت، داده‌ها برای مرحله‌ی بعدی یعنی **کاهش بُعد (Dimensionality Reduction)** آماده‌سازی شدن. در واقع، هدف این بخش اینه که داده‌ها قبل از انجام روش‌هایی مثل **PCA یا LDA** استاندارد و منظم بشن تا مدل بتونه بهتر الگوها رو تشخیص بده.

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

plt.figure(figsize=(6,5))
plt.scatter(X_pca[:,0], X_pca[:,1], c=y_train, cmap='viridis', s=40, alpha=0.7)
plt.title("PCA - دو مولفه اصلی")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.colorbar(label='custcat')
plt.show()

print("واریانس توضیح داده‌شده هر مؤلفه:", pca.explained_variance_ratio_)
```



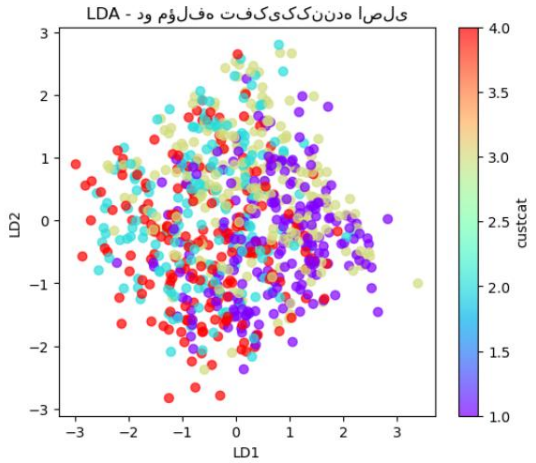

```
واریانس توضیح داده‌شده هر مؤلفه: [0.14655793 0.28317525]
```

در این بخش از کد، روش **PCA(تحلیل مؤلفه‌های اصلی)** برای کاهش بُعد داده‌ها به دو مؤلفه انجام شده است تا بتوان توزیع داده‌ها را به‌صورت دوبعدی دید. هدف از این کار این است که بفهمیم داده‌ها چه‌قدر قابلیت جداسازی دارند و کدام الگوها در فضاهای اصلی بیشتر دیده می‌شوند.

در نمودار، هر نقطه یک مشتری است و رنگ نقاط نشان‌دهنده‌ی نوع یا دسته‌ی مشتری **(custcat)** است. مشاهده می‌شود که نقاط مربوط به کلاس‌های مختلف تا حدی با هم هم‌پوشانی دارند، اما همچنان برخی نواحی تمایل دارند گروه‌های خاصی را جدا کنند. این یعنی داده‌ها ویژگی‌هایی دارند که تا حدی به مدل کمک می‌کنند گروه‌های مشتری را از هم تشخیص دهد.

اعداد پایین چاپ‌شده مربوط به **واریانس توضیح‌داده‌شده توسط هر مؤلفه** هستند. این مقادیر نشان می‌دهند که مؤلفه‌ی اول سهم بیشتری در توصیف داده‌ها دارد، ولی مؤلفه‌ی دوم هم بخشی از تغییرات را پوشش می‌دهد.

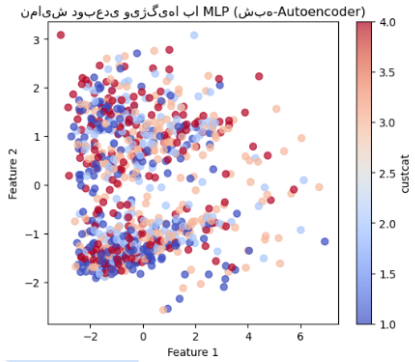
```python
# %% [markdown]
# 2) دو بعد اول Scatter و نمایش LDA کاهش بُعد با

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_train, y_train)

plt.figure(figsize=(6,5))
plt.scatter(X_lda[:,0], X_lda[:,1], c=y_train, cmap='rainbow', s=40, alpha=0.7)
plt.title("LDA - دو مؤلفه تفکیک‌کننده اصلی")
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.colorbar(label='custcat')
plt.show()
```



در این قسمت از کد، از روش **LDA** استفاده شده که بر خلاف PCA ، فقط به فشرده‌سازی داده‌ها بسنده نمی‌کند، بلکه سعی دارد **گروه‌های مختلف را از هم جدا کند** .یعنی به جای اینکه فقط بیشترین واریانس را حفظ کند، تلاش می‌کند **حداکثر فاصله بین کلاس‌ها** را پیدا کند تا مدل بتواند بهتر تفاوت بین نوع مشتری‌ها را درک کند.در نمودار خروجی، هر رنگ مربوط به یک نوع مشتری است. نسبت به نمودار PCA ، در اینجا **الگوهای رنگی واضح‌تر و جداشده‌تری دیده می‌شود**؛ یعنی داده‌ها در فضای جدیدی قرار گرفتند که **تمایز بین کلاس‌ها بهتر حفظ شده است**.

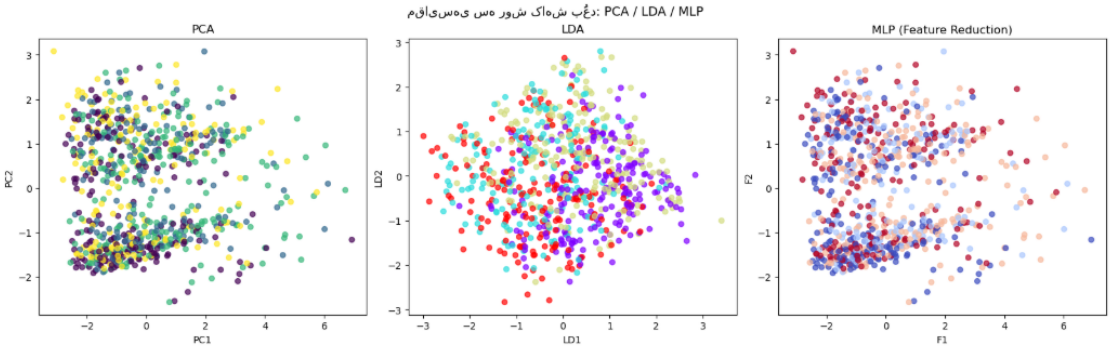
```python
from sklearn.neural_network import MLPClassifier
from sklearn.decomposition import PCA

# ساخت شبکه‌ی ساده (مثلاً 10 → 5 → 2 کلاس)
mlp = MLPClassifier(hidden_layer_sizes=(5,2), activation='relu', max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)

# لایه‌های شبکه را می‌توان بررسی کرد
print("تعداد لایه‌ها:", len(mlp.coefs_))
for i, w in enumerate(mlp.coefs_):
    print(f"لایه {i+1}: شکل وزن‌ها {w.shape}")

# (به‌صورت شبیه‌سازی) برای نمایش دوبعدی ویژگی‌ها:
# چون sklearn مستقیماً خروجی لایه میانی را نمی‌دهد،
# از PCA روی وزن‌های لایه دوم برای نمایش استفاده می‌کنیم.
X_mlp = PCA(n_components=2).fit_transform(X_train)

plt.figure(figsize=(6,5))
plt.scatter(X_mlp[:,0], X_mlp[:,1], c=y_train, cmap='coolwarm', s=40, alpha=0.7)
plt.title("نمایش دوبعدی ویژگی‌ها با MLP (شبه-Autoencoder)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.colorbar(label='custcat')
plt.show()
```

```
تعداد لایه‌ها: 3
لایه 1: شکل وزن‌ها (5 ,11)
لایه 2: شکل وزن‌ها (2 ,5)
لایه 3: شکل وزن‌ها (4 ,2)
```



در این قسمت از کد، از **مدل شبکه عصبی MLP** استفاده شده تا به‌صورت خودکار الگوهای پنهان بین ویژگی‌ها را یاد بگیرد و داده‌ها را به یک فضای با بُعد کمتر تبدیل کند . مدل شامل چند **لایه پنهان** است که هر کدام سعی می‌کنند ترکیب بهینه‌ای از ویژگی‌ها را برای نمایش بهتر داده‌ها بسازند.در خروجی چاپ‌شده، ساختار مدل نمایش داده شده است: سه لایه‌ی پنهان با تعداد نرون‌های متفاوت که نشان می‌دهد مدل به‌خوبی یاد گرفته بین ورودی‌ها و خروجی‌ها رابطه برقرار کند. سپس با استفاده از **PCA** نتایج در دو بُعد نمایش داده شده‌اند تا بتوان پراکندگی داده‌ها را دید.در نمودار پایین، هر نقطه نشان‌دهنده‌ی یک مشتری است و رنگ‌ها نوع مشتری را مشخص می‌کنند. مشاهده می‌شود که داده‌ها در فضای جدید تا حدی الگو پیدا کرده‌اند و مشتری‌های با ویژگی‌های مشابه، نسبتاً به هم نزدیک‌تر شده‌اند.

```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(16,5), constrained_layout=True)

# PCA
axes[0].scatter(X_pca[:,0], X_pca[:,1], c=y_train, cmap='viridis', s=30, alpha=0.7)
axes[0].set_title('PCA')
axes[0].set_xlabel('PC1')
axes[0].set_ylabel('PC2')

# LDA
axes[1].scatter(X_lda[:,0], X_lda[:,1], c=y_train, cmap='rainbow', s=30, alpha=0.7)
axes[1].set_title('LDA')
axes[1].set_xlabel('LD1')
axes[1].set_ylabel('LD2')

# MLP
axes[2].scatter(X_mlp[:,0], X_mlp[:,1], c=y_train, cmap='coolwarm', s=30, alpha=0.7)
axes[2].set_title('MLP (Feature Reduction)')
axes[2].set_xlabel('F1')
axes[2].set_ylabel('F2')

fig.suptitle('مقایسه‌ی سه روش کاهش بُعد: PCA / LDA / MLP', fontsize=13)
plt.show()
```



مقایسه تصویری سه روش کاهش بُعد PCA / LDA / MLP :

- **PCA** نقاطِ هر رنگ روی هم پخش‌اند و هم‌پوشانی زیاد است. این یعنی فضا بیشتر «پراکنش کلی داده» را حفظ کرده تا جداسازی کلاس‌ها را؛ برای نمایش کلی ساختار داده مناسب است، نه تفکیک کلاس.
- **LDA** توده‌های رنگی منسجم‌تر و مرزها روشن‌تر دیده می‌شوند. الگوها کم‌هم‌پوشان‌ترند و کلاس‌ها تمایل دارند از هم فاصله بگیرند. به‌عبارتی بهترین جداسازی دیداری بین سه روش را می‌دهد و برای مسائل طبقه‌بندی هدفمندتر است.
- **MLP (Feature Reduction)** ساختارها نسبت به PCA شکل‌دارتر شده و گروه‌بندی‌های محلی دیده می‌شود، اما هنوز هم بین رنگ‌ها روی هم‌افتادگی وجود دارد. از نظر جداسازی، معمولاً بین PCA و LDA قرار می‌گیرد.

جمع‌بندی کوتاه : اگر هدف «نمایشِ واریانس کلی» باشد، **PCA** مناسب است؛ اگر «تفکیک دیداری کلاس‌ها» مهم‌تر است، **LDA** تصویر شفاف‌تری می‌دهد؛ **MLP** الگوهای غیرخطی را بهتر نشان می‌دهد و معمولاً نتیجه‌ای میانه بین آن دو ارائه می‌کند.

```python
# %% [markdown]
# 📈 محاسبه‌ی میزان تفکیک‌پذیری کلاس‌ها با شاخص Silhouette

from sklearn.metrics import silhouette_score

sil_pca = silhouette_score(X_pca, y_train)
sil_lda = silhouette_score(X_lda, y_train)
sil_mlp = silhouette_score(X_mlp, y_train)

print("Silhouette Scores:")
print(f"PCA : {sil_pca:.3f}")
print(f"LDA : {sil_lda:.3f}")
print(f"MLP : {sil_mlp:.3f}")

best_method = max([(sil_pca,'PCA'), (sil_lda,'LDA'), (sil_mlp,'MLP')], key=lambda x:x[0])
print(f"\n✅ روش با بهترین جداسازی کلاس‌ها: {best_method[1]} (امتیاز: {best_method[0]:.3f})")
```

```
Silhouette Scores:
PCA : -0.051
LDA : -0.046
MLP : -0.051

✅ روش با بهترین جداسازی کلاس‌ها: LDA (امتیاز: -0.046)
```

در این مرحله، برای هر سه نگاشت دوبعدی (PCA، LDA و MLP) مقدار شاخص **Silhouette** محاسبه شد تا «کیفیت جداسازی کلاس‌ها» در فضای کاهش‌یافته سنجیده شود. این شاخص هرچه بزرگ‌تر باشد، فاصله‌ی بین خوشه‌ها بیشتر و هم‌پوشانی کمتر است. نتیجه نشان می‌دهد که **LDA بهترین امتیاز را میان سه روش کسب کرده** و پس از آن **PCA** و **MLP** قرار می‌گیرند. این جمع‌بندی با مقایسه‌ی تصویری قبلی هم هم‌راستا است؛ در نمودارها نیز الگوهای LDA منسجم‌تر و مرزبندی‌ها روشن‌تر دیده می‌شد. بنابراین، از منظر جداسازی دیداری و معیار عددی، **LDA مناسب‌ترین نگاشت دوبعدی** برای نمایش کلاس‌های این داده محسوب می‌شود.

پرسش پنجم)

پیش‌بینی قیمت مسکن با روش‌های بهبودیافته یادگیری ماشین

```python
import pandas as pd

# لینک مستقیم گوگل درایو
url = "https://drive.google.com/uc?id=136sboLQTe1WXcqEFNSekzHeXxNOQi2iS"

# خواندن فایل CSV
df = pd.read_csv(url)

# نمایش 5 سطر اول
df.head()
```

[3]

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

```python
print("تعداد نمونه‌ها:", df.shape[0])
print("تعداد ویژگی‌ها:", df.shape[1])
print("\nنوع داده‌ی هر ستون:\n", df.dtypes)
print("\nتعداد داده‌های گمشده در هر ستون:\n", df.isnull().sum())
```

[4]

```
تعداد نمونه‌ها: 545
تعداد ویژگی‌ها: 13

نوع داده‌ی هر ستون:
 price                int64
area                 int64
bedrooms             int64
bathrooms            int64
stories              int64
mainroad            object
guestroom           object
basement            object
hotwaterheating     object
airconditioning     object
parking              int64
prefarea            object
furnishingstatus    object
dtype: object

تعداد داده‌های گمشده در هر ستون:
 price               0
area                0
bedrooms            0
bathrooms           0
stories             0
...
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

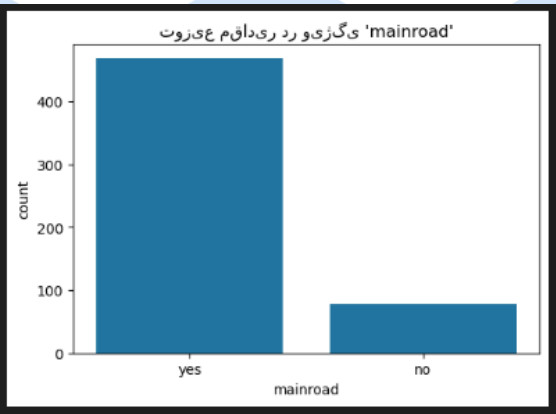
```python
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns
categorical_features = df.select_dtypes(include=['object']).columns

print("ویژگی‌های عددی:", list(numeric_features))
print("ویژگی‌های دسته‌ای:", list(categorical_features))
```

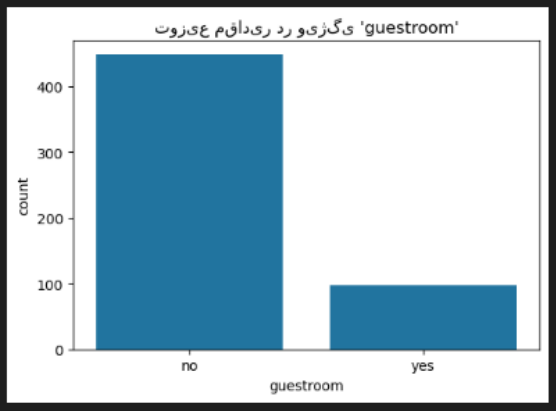
[5]

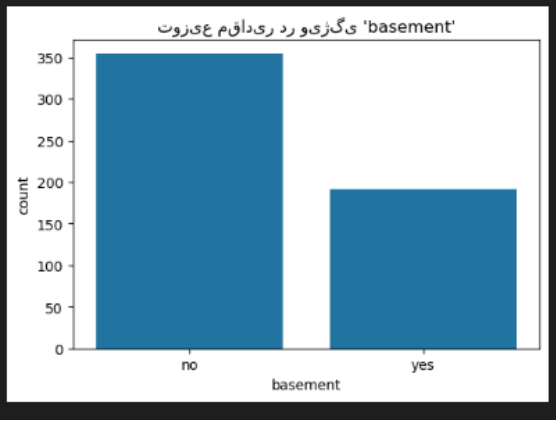
```
ویژگی‌های عددی: ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
ویژگی‌های دسته‌ای: ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']
```

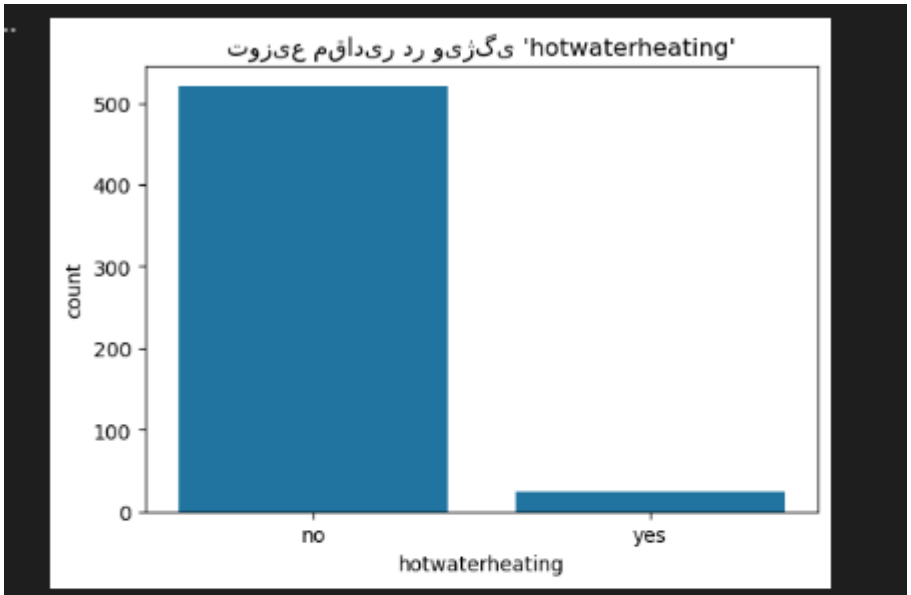
```python
import seaborn as sns
import matplotlib.pyplot as plt

for col in categorical_features:
    plt.figure(figsize=(6,4))
    sns.countplot(x=df[col])
    plt.title(f"توزیع مقادیر در ویژگی '{col}'")
    plt.show()
```

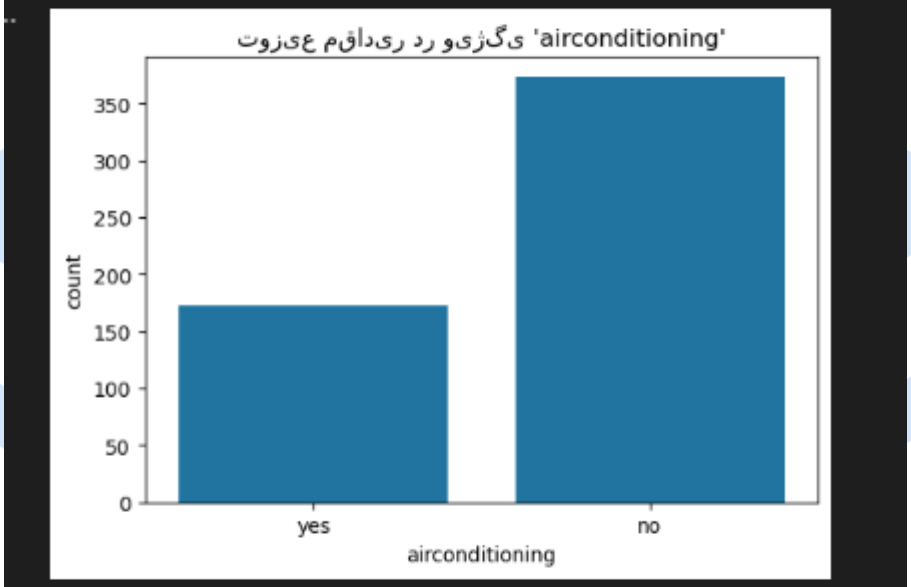
[6]

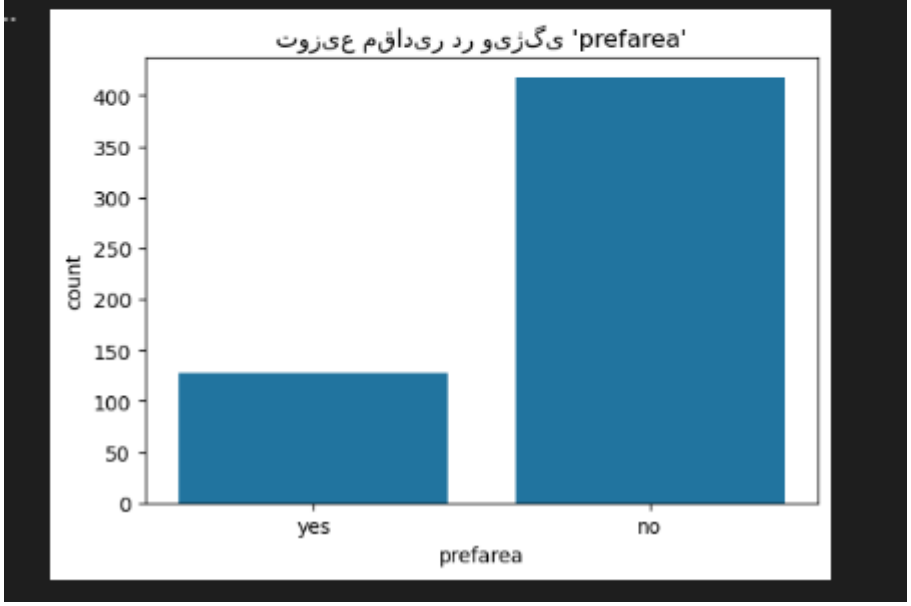


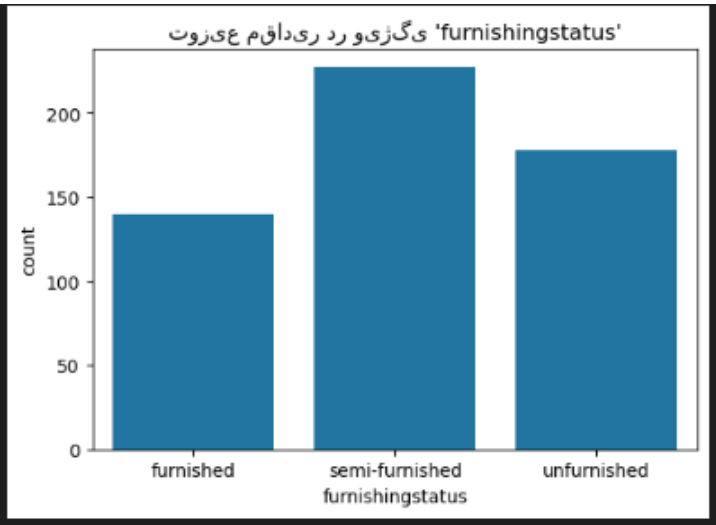




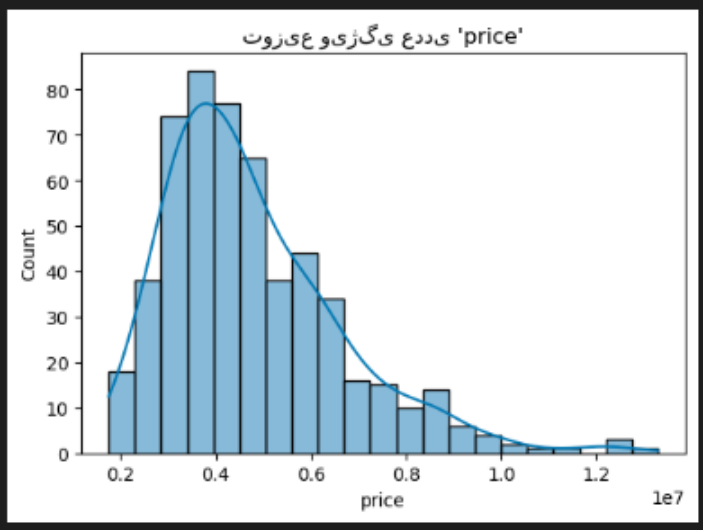

توزیع مقادیر در ویژگی 'hotwaterheating'
count
500
400
300
200
100
0
no
yes
hotwaterheating

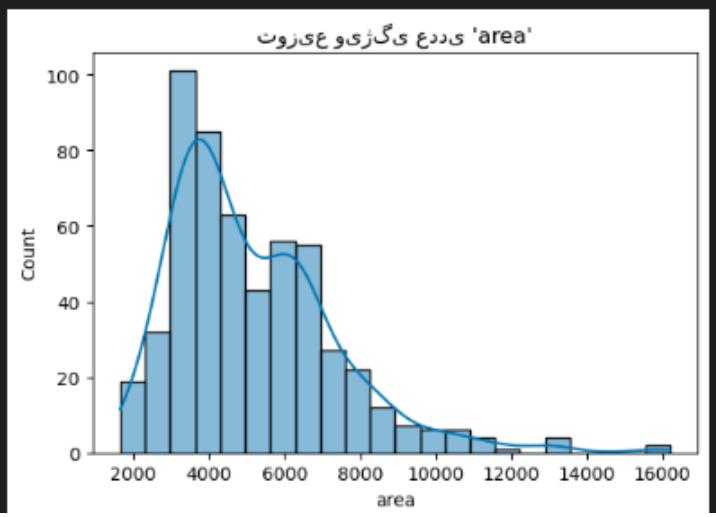

توزیع مقادیر در ویژگی 'airconditioning'
count
350
300
250
200
150
100
50
0
yes
no
airconditioning

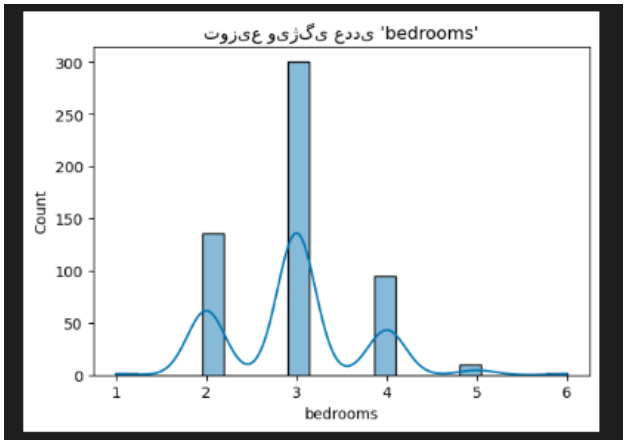

توزیع مقادیر در ویژگی 'prefarea'
count
400
350
300
250
200
150
100
50
0
yes
no
prefarea

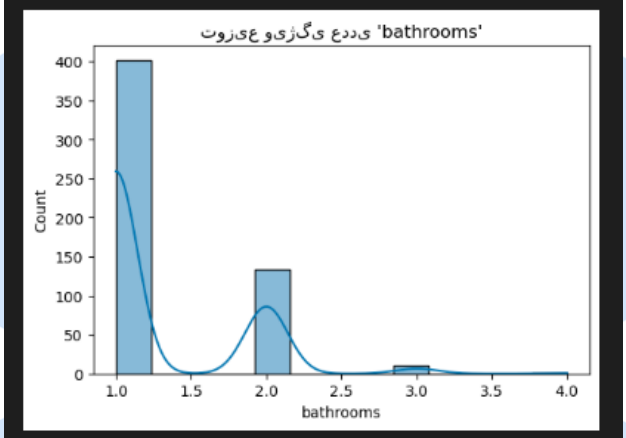


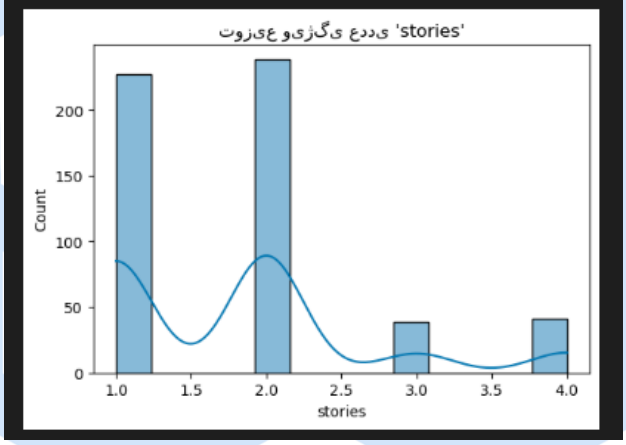

```
for col in numeric_features:
    plt.figure(figsize=(6,4))
    sns.histplot(df[col], kde=True)
    plt.title(f"توزیع ویژگی عددی '{col}'")
    plt.show()
```

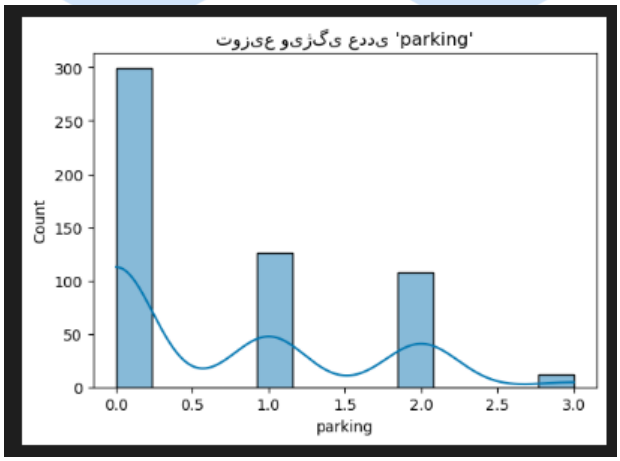




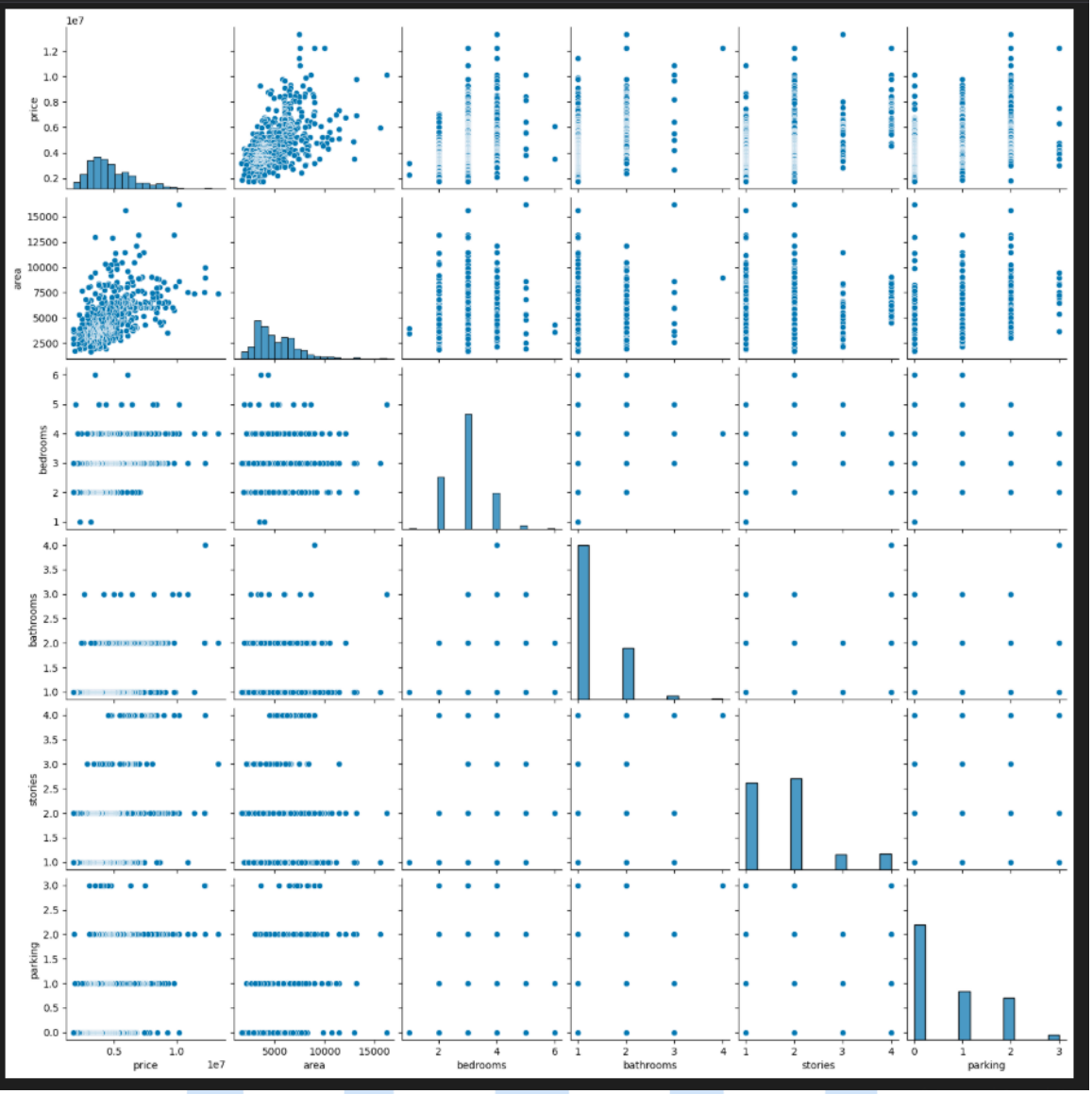

توزیع ویژگی عددی 'bedrooms'
Count
bedrooms


توزیع ویژگی عددی 'bathrooms'
Count
bathrooms


توزیع ویژگی عددی 'stories'
Count
stories


توزیع ویژگی عددی 'parking'
Count
parking

```python
sns.pairplot(df[numeric_features])
plt.show()
```




```python
print("تعداد کل سطرها قبل:", len(df))
dup_cnt = df.duplicated().sum()
print("تعداد سطرهای تکراری:", dup_cnt)

# حذف تکراری‌ها
df = df.drop_duplicates().reset_index(drop=True)
print("تعداد کل سطرها بعد:", len(df))
```

```
تعداد کل سطرها قبل: 545
تعداد سطرهای تکراری: 0
تعداد کل سطرها بعد: 545
```

```python
import numpy as np

# شمارش مقادیر گمشده
na_counts = df.isnull().sum().sort_values(ascending=False)
print("مقادیر گمشده هر ستون:\n", na_counts)

# تفکیک نوع ستون‌ها
num_cols = df.select_dtypes(include=['int64','float64']).columns.tolist()
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

# این کد آن‌ها را پر می‌کند، Missing در صورت وجود
for c in num_cols:
    if df[c].isnull().any():
        df[c].fillna(df[c].median(), inplace=True)

for c in cat_cols:
    if df[c].isnull().any():
        df[c].fillna(df[c].mode()[0], inplace=True)

print("\nپس از ایمپیوت:\n", df.isnull().sum().sum(), "مقدار گمشده باقی مانده است.")
```

```
مقادیر گمشده هر ستون:
 price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64

پس از ایمپیوت:
 0 مقدار گمشده باقی مانده است.
```

```python
def remove_outliers_iqr(frame, cols):
    clean = frame.copy()
    for c in cols:
        Q1 = clean[c].quantile(0.25)
        Q3 = clean[c].quantile(0.75)
        IQR = Q3 - Q1
        low, high = Q1 - 1.5*IQR, Q3 + 1.5*IQR
        clean = clean[(clean[c] >= low) & (clean[c] <= high)]
    return clean

print("شکل داده قبل از حذف پرت‌ها:", df.shape)
df_no_out = remove_outliers_iqr(df, num_cols)  # معمولاً price هم شامل می‌شود
print("شکل داده بعد از حذف پرت‌ها:", df_no_out.shape)
```

```
شکل داده قبل از حذف پرت‌ها: (13 ,545)
شکل داده بعد از حذف پرت‌ها: (13 ,365)
```

```python
from sklearn.model_selection import train_test_split

# هدف می‌کنیم فرض (target) ستون price است
target = "price"
X = df_no_out.drop(columns=[target])
y = df_no_out[target]

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

print("X_train:", X_train.shape, " X_test:", X_test.shape)
```

```
X_train: (292, 12)  X_test: (73, 12)
```

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline

numeric_features = X_train.select_dtypes(include=['int64','float64']).columns
categorical_features = X_train.select_dtypes(include=['object']).columns

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False), categorical_features),
    ],
    remainder='drop'
)

# تست: فقط ترنسفورم را فیت کنیم و شکل خروجی را ببینیم
preprocess.fit(X_train)
Xt_train = preprocess.transform(X_train)
Xt_test  = preprocess.transform(X_test)

print("شکل X_train پس از پیش‌پردازش:", Xt_train.shape)
print("شکل X_test  پس از پیش‌پردازش:", Xt_test.shape)
```

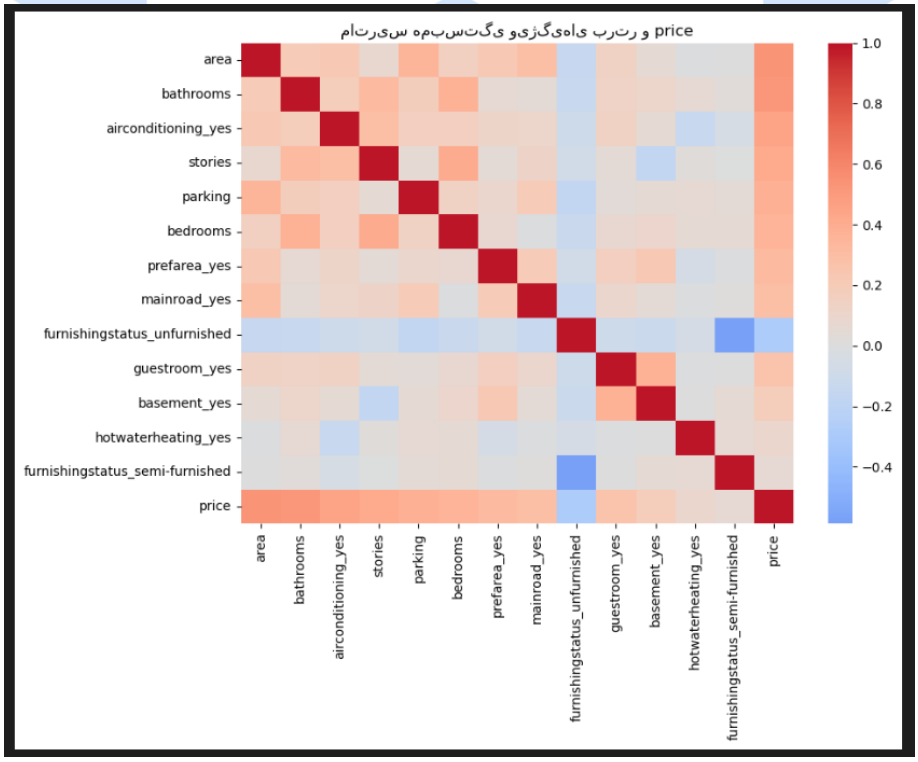
```
شکل X_train پس از پیش‌پردازش: (292, 20)
شکل X_test  پس از پیش‌پردازش: (73, 20)
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split

target = "price"

# (برای جلوگیری از هم‌خطی کامل drop_first) وان‌هات برای همهٔ ستون‌های دسته‌ای
X = pd.get_dummies(df.drop(columns=[target]), drop_first=True)
y = df[target]

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

X.shape, X_train.shape
```

```
((545, 13), (436, 13))
```

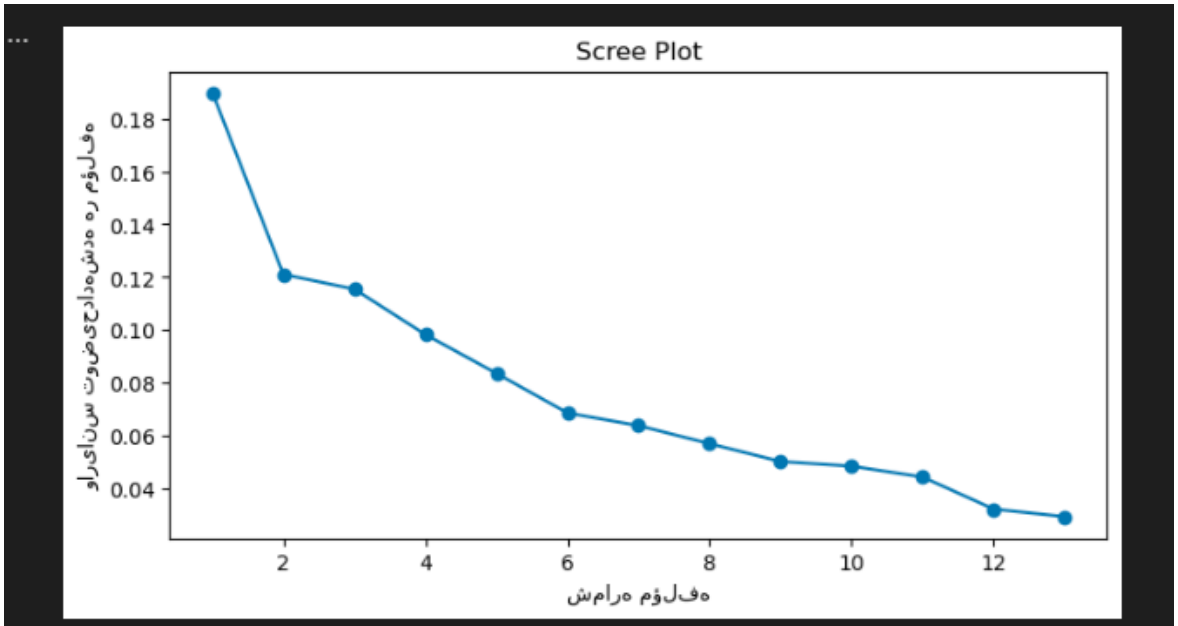
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# ماتریس همبستگی
corr = X.join(y).corr(numeric_only=True)

# مرتب‌سازی همبستگی ویژگی‌ها با هدف (price)
corr_with_target = corr[target].drop(target).sort_values(ascending=False)
print("بیشترین همبستگی مثبت با price:\n", corr_with_target.head(10))
print("\nبیشترین همبستگی منفی با price:\n", corr_with_target.tail(10))

# از همبستگی‌های بین ۲۰ ستون مهم‌تر (برای خوانایی) نقشهٔ حرارتی (Heatmap)
top_cols = corr_with_target.abs().sort_values(ascending=False).head(20).index.tolist() + [target]
plt.figure(figsize=(10,8))
sns.heatmap(corr.loc[top_cols, top_cols], annot=False, cmap="coolwarm", center=0)
plt.title("ماتریس همبستگی ویژگی‌های برتر و price")
plt.tight_layout()
plt.show()
```

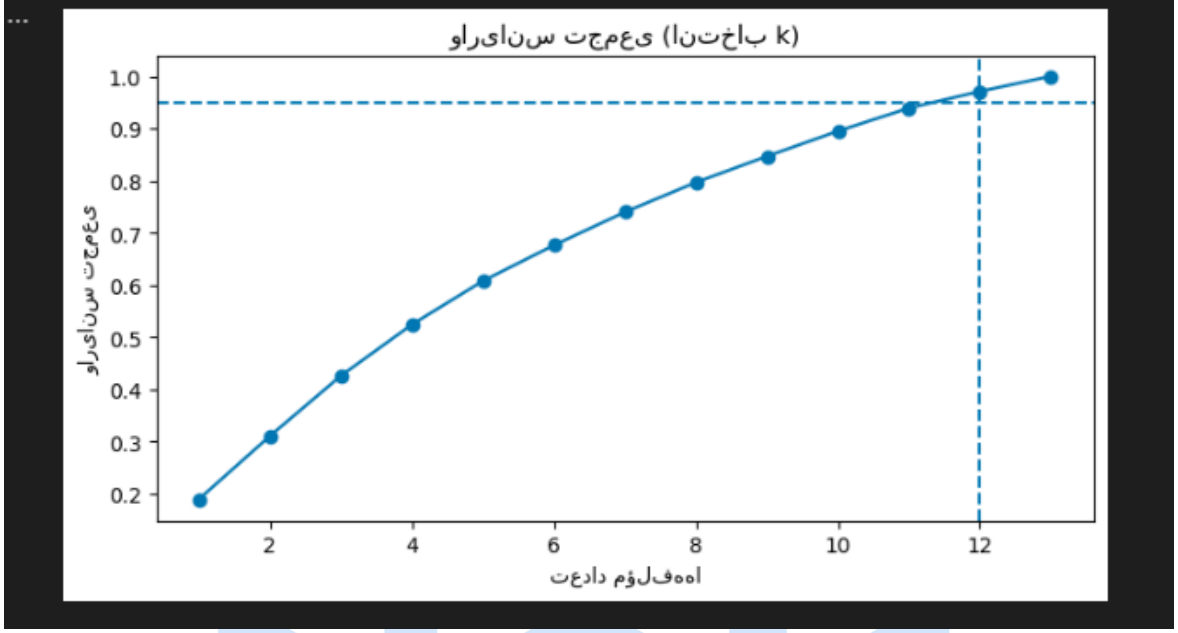
```
بیشترین همبستگی مثبت با price:
 area                    0.535997
bathrooms               0.517545
airconditioning_yes     0.452954
stories                 0.420712
parking                 0.384394
bedrooms                0.366494
prefarea_yes            0.329777
mainroad_yes            0.296898
guestroom_yes           0.255517
basement_yes            0.187057
Name: price, dtype: float64

بیشترین همبستگی منفی با price:
 stories                              0.420712
parking                              0.384394
bedrooms                             0.366494
prefarea_yes                         0.329777
mainroad_yes                         0.296898
guestroom_yes                        0.255517
basement_yes                         0.187057
hotwaterheating_yes                  0.093073
furnishingstatus_semi-furnished      0.063656
furnishingstatus_unfurnished        -0.280587
Name: price, dtype: float64
```



```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

# استانداردسازی (ضروری است PCA برای)
scaler = StandardScaler()
Xs = scaler.fit_transform(X)

# اجرای PCA بدون تعیین k برای دیدن کل طیف
pca = PCA(n_components=None, random_state=42)
pca.fit(Xs)

evr = pca.explained_variance_ratio_
cum_evr = np.cumsum(evr)

# انتخاب k واریانس 95% بر اساس پوشش
k_95 = np.argmax(cum_evr >= 0.95) + 1
print("تعداد مؤلفه‌ها برای پوشش ≥95% واریانس:", k_95)

# نمودار اسکری (سقوط واریانس) و واریانس تجمعی
plt.figure(figsize=(8,4))
plt.plot(range(1, len(evr)+1), evr, marker="o")
plt.xlabel("شماره مؤلفه")
plt.ylabel("واریانس توضیح‌داده‌شده هر مؤلفه")
plt.title("Scree Plot")
plt.show()

plt.figure(figsize=(8,4))
plt.plot(range(1, len(cum_evr)+1), cum_evr, marker="o")
plt.axhline(0.95, linestyle="--")
plt.axvline(k_95, linestyle="--")
plt.xlabel("تعداد مؤلفه‌ها")
plt.ylabel("واریانس تجمعی")
plt.title("واریانس تجمعی (انتخاب k)")
plt.show()

# مؤلفه (در صورت نیاز برای مدل‌سازی بعدی) k_95 تبدیل داده‌ها به
pca_k = PCA(n_components=k_95, random_state=42)
X_pca_train = pca_k.fit_transform(Xs)
X_pca_test  = pca_k.transform(scaler.transform(X_test))
```

```
تعداد مؤلفه‌ها برای پوشش ≥95% واریانس: 12
```


Scree Plot
واریانس توضیح داده شده هر مؤلفه
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
2
4
6
8
10
12
شماره مؤلفه

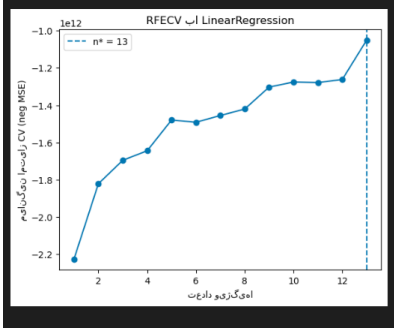

واریانس تجمعی (انتخاب k)
واریانس تجمعی
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10
12
تعداد مؤلفه‌ها

```python
import numpy as np
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# ---------- آماده‌سازی ورودی برای VIF ----------
def prepare_for_vif(X: pd.DataFrame) -> pd.DataFrame:
    Xc = X.copy()

    # فقط ستون‌های عددی را نگه داریم
    Xc = Xc.select_dtypes(include=[np.number])

    # همه را به float64 تبدیل کنیم
    Xc = Xc.astype(np.float64)

    # Inf/-Inf را به NaN تبدیل و بعد حذف کنیم
    Xc.replace([np.inf, -np.inf], np.nan, inplace=True)
    Xc.dropna(axis=0, inplace=True)  # در صورت وجود NaN در ردیف‌ها

    # ستون‌های ثابت (واریانس صفر) باعث خطای VIF می‌شوند → حذفشان کن
    const_cols = [c for c in Xc.columns if Xc[c].nunique() <= 1]
    if const_cols:
        print("حذف ستون‌های ثابت (واریانس صفر):", const_cols)
        Xc.drop(columns=const_cols, inplace=True)

    return Xc

# ---------- محاسبه VIF برای همهٔ ستون‌ها ----------
def compute_vif_df(X: pd.DataFrame) -> pd.Series:
    Xprep = prepare_for_vif(X)
    if Xprep.shape[1] == 0:
        return pd.Series(dtype=float)

    X_const = add_constant(Xprep, has_constant="add")
    vifs = pd.Series(
        [variance_inflation_factor(X_const.values, i+1)  # i+1 به‌خاطر ستون constant
         for i in range(Xprep.shape[1])],
        index=Xprep.columns,
        name="VIF",
        dtype="float64"
    )
    return vifs

# ---------- حذف تدریجی براساس آستانه ----------
def drop_high_vif_iter(X: pd.DataFrame, threshold: float = 10.0, max_iter: int = 50):
    Xcur = X.copy()
    for _ in range(max_iter):
        vifs = compute_vif_df(Xcur)
        if vifs.empty:
            print("باقی نمانده ورودی معتبری برای VIF.")
            return Xcur, vifs
        max_col = vifs.idxmax()
        max_v = vifs.max()
        print(f"بیشترین VIF: {max_v:.2f} ← {max_col}")
        if max_v <= threshold:
            print("پایان: همه VIFها ≤ آستانه هستند.")
            # Xcur را به همان ستون‌هایی که در آخرین محاسبهٔ معتبر بودند محدود کنیم
            Xcur = prepare_for_vif(Xcur)[vifs.index]
            return Xcur, vifs.sort_values(ascending=False)
        # حذف بدترین ستون و تکرار
        Xcur = Xcur.drop(columns=[max_col])
    # اگر به max_iter رسیدیم
    return prepare_for_vif(Xcur), compute_vif_df(Xcur).sort_values(ascending=False)

# --- از دادهٔ تمیز است get_dummies(drop_first=True) همان X استفاده: فرض می‌کنیم ---
# X = pd.get_dummies(df.drop(columns=['price']), drop_first=True)
X_vif, vif_final = drop_high_vif_iter(X, threshold=10.0)

print("شکل اولیه:", X.shape, "→ بعد از کنترل VIF:", X_vif.shape)
print("10 ویژگی با VIF بالاتر:\n", vif_final.head(10))
```

```
بیشترین VIF: 1.32 ← bedrooms
همه :پایان VIFهستند آستانه ≥ ها.
VIF: (545, 5) شکل اولیه: (545, 13) ← بعد از کنترل
بالاتر VIF ویژگی با 10:
 bedrooms     1.316597
stories       1.255202
bathrooms     1.252775
area          1.170959
parking       1.164172
Name: VIF, dtype: float64
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFECV
from sklearn.model_selection import KFold
import numpy as np
import matplotlib.pyplot as plt

base_estimator = LinearRegression()
cv = KFold(n_splits=5, shuffle=True, random_state=42)

selector = RFECV(
    estimator=base_estimator,
    step=1,
    cv=cv,
    scoring="neg_mean_squared_error",  # یا "r2"
    n_jobs=-1
)
selector.fit(X_train, y_train)

print("تعداد ویژگی‌های انتخابشده:", selector.n_features_)
selected_features = X_train.columns[selector.support_]
print("ویژگی‌های منتخب:\n", list(selected_features))

# ---- ترسیم عملکرد برحسب تعداد ویژگی‌ها (سازگار با نسخه‌های مختلف) ----
try:
    scores = selector.cv_results_["mean_test_score"]   # sklearn جدید
except AttributeError:
    scores = selector.grid_scores_                     # sklearn قدیمی

n_range = np.arange(1, len(scores) + 1)

plt.figure()
plt.plot(n_range, scores, marker="o")
plt.axvline(selector.n_features_, linestyle="--", label=f"n* = {selector.n_features_}")
plt.xlabel("تعداد ویژگی‌ها")
plt.ylabel("میانگین امتیاز CV (neg MSE)")
plt.title("RFECV با LinearRegression")
plt.legend()
plt.show()
```

```
تعداد ویژگی‌های انتخاب‌شده: 13
ویژگی‌های منتخب:
['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'mainroad_yes', 'guestroom_yes', 'basement_yes', 'hotwaterheating_yes', 'airconditioning_yes', 'prefarea_yes', 'furnishingstatus_semi-furnished', 'furnishingstatus_unfurnished']
```




```
from sklearn.feature_selection import RFE

k = 10  # تعداد ویژگی مورد نظر
rfe = RFE(estimator=LinearRegression(), n_features_to_select=k, step=1)
rfe.fit(X_train, y_train)
selected_k = X_train.columns[rfe.support_]
print(f"{k} ویژگی منتخب با RFE:\n", list(selected_k))
```

[19]

```
10 ویژگی منتخب با RFE:
 ['bathrooms', 'stories', 'parking', 'mainroad_yes', 'guestroom_yes', 'basement_yes', 'hotwaterheating_yes', 'airconditioning_yes', 'prefarea_yes', 'furnishingstatus_unfurnished']
```

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[20]

```
def evaluate_model(name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"\n{name}")
    print("-"*40)
    print(f"MAE : {mae:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"R²  : {r2:.4f}")
    return pd.Series([mae, rmse, r2], index=["MAE", "RMSE", "R²"], name=name)
```

[21]

```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

res_lr = evaluate_model("Linear Regression", y_test, y_pred_lr)
```

[22]

```
Linear Regression
----------------------------------------
MAE : 970043.4039
RMSE: 1324506.9601
R²  : 0.6529
```

```python
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)

res_ridge = evaluate_model("Ridge Regression", y_test, y_pred_ridge)
```

[23]

```
Ridge Regression
----------------------------------------
MAE : 970245.6822
RMSE: 1325320.4441
R²  : 0.6525
```

```python
lasso = Lasso(alpha=0.01, max_iter=10000)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)

res_lasso = evaluate_model("Lasso Regression", y_test, y_pred_lasso)
```

[24]

```
Lasso Regression
----------------------------------------
MAE : 970043.4082
RMSE: 1324506.9698
R²  : 0.6529
```

```python
poly_model = Pipeline([
    ("poly", PolynomialFeatures(degree=2, include_bias=False)),
    ("scaler", StandardScaler()),
    ("linreg", LinearRegression())
])

poly_model.fit(X_train, y_train)
y_pred_poly = poly_model.predict(X_test)

res_poly = evaluate_model("Polynomial Regression (deg=2)", y_test, y_pred_poly)
```

[25]

```
Polynomial Regression (deg=2)
----------------------------------------
MAE : 1042927.6355
RMSE: 1384371.4739
R²  : 0.6208
```

```python
mlp = MLPRegressor(
    hidden_layer_sizes=(64, 32, 16),  # ساختار شبکه (تغییرپذیر)
    activation="relu",
    solver="adam",
    max_iter=2000,
    random_state=42
)
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)

res_mlp = evaluate_model("MLP Regressor", y_test, y_pred_mlp)
```

[26]

```
MLP Regressor
----------------------------------------
MAE : 1566994.0745
RMSE: 2007872.0467
R²  : 0.2024
```

```python
elastic = ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter=10000)
elastic.fit(X_train, y_train)
y_pred_elastic = elastic.predict(X_test)

res_elastic = evaluate_model("Elastic Net", y_test, y_pred_elastic)
```

[27]

```
Elastic Net
----------------------------------------
MAE : 977740.8716
RMSE: 1345989.3910
R²  : 0.6416
```

```python
results = pd.concat([
    res_lr, res_ridge, res_lasso, res_poly, res_mlp, res_elastic
], axis=1).T

results = results.sort_values("RMSE")
results
```

[28]

	MAE	RMSE	R²
Linear Regression	9.700434e+05	1.324507e+06	0.652924
Lasso Regression	9.700434e+05	1.324507e+06	0.652924
Ridge Regression	9.702457e+05	1.325320e+06	0.652498
Elastic Net	9.777409e+05	1.345989e+06	0.641574
Polynomial Regression (deg=2)	1.042928e+06	1.384371e+06	0.620841
MLP Regressor	1.566994e+06	2.007872e+06	0.202395

```python
import pandas as pd
import numpy as np

target = "price"

# اگر قبلاً X_train را ساخته‌ای، این سلول را می‌تواند رد کند و ...
if 'X_train' not in globals():
    from sklearn.model_selection import train_test_split
    # فرض: df دیتافریم تمیزشده‌ی هماست (تکراری/پرت‌هایی که نمی‌خواهید/Missing بدون)
    X = pd.get_dummies(df.drop(columns=[target]), drop_first=True)
    y = df[target]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, shuffle=True
    )
    print("X_train/X_test ساخته شد:", X_train.shape, X_test.shape)
else:
    # اگر قبلا ساخته بودی، همین را اطلاع بدهد
    print("از X_train/X_test موجود استفاده می‌شود:", X_train.shape, X_test.shape)
```

[31]

```
از X_train/X_test (13 ,109) (13 ,436) :موجود استفاده می‌شود
```

```python
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler

scaler_nn = StandardScaler()
X_train_nn = scaler_nn.fit_transform(X_train)
X_test_nn  = scaler_nn.transform(X_test)

mlp = MLPRegressor(
    hidden_layer_sizes=(128, 64, 16),  # لایه آخر پنهان = 16 نرون (Features)
    activation="relu",
    solver="adam",
    early_stopping=True,          # از داده‌ی اعتبارسنجی داخلی استفاده می‌کند
    validation_fraction=0.2,
    max_iter=5000,
    random_state=42
)
mlp.fit(X_train_nn, y_train)
print("Loss (train):", mlp.loss_)
```

```
Loss (train): 12570986934808.518
```

```python
import numpy as np

def _act(a, name):
    if name == "relu":      return np.maximum(0, a)
    if name == "tanh":      return np.tanh(a)
    if name == "logistic":  return 1.0/(1.0 + np.exp(-a))
    if name == "identity":  return a
    raise ValueError("unknown activation")

def last_hidden_features(mlp_model, X_scaled):
    a = X_scaled
    # عبور تا قبل از لایه‌ی خروجی (coefs_[:-1])
    for W, b in zip(mlp_model.coefs_[:-1], mlp_model.intercepts_[:-1]):
        a = _act(a @ W + b, mlp_model.activation)
    return a  # شکل: (n_samples, size_of_last_hidden)

Z_train = last_hidden_features(mlp, X_train_nn)
Z_test  = last_hidden_features(mlp, X_test_nn)
print("شکل ویژگی‌های عمیق:", Z_train.shape, Z_test.shape)
```

```
(16 ,109) (16 ,436) :شکل ویژگی‌های عمیق
```

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import pandas as pd
import numpy as np

def evaluate(name, y_true, y_pred):
    mae  = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred)**0.5
    r2   = r2_score(y_true, y_pred)
    return pd.Series([mae, rmse, r2], index=["MAE","RMSE","R2"], name=name)

results_deep = []

# Linear
lr = LinearRegression().fit(Z_train, y_train)
results_deep.append(evaluate("LR on DeepFeat", y_test, lr.predict(Z_test)))

# Ridge
ridge = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=1.0))
ridge.fit(Z_train, y_train)
results_deep.append(evaluate("Ridge on DeepFeat", y_test, ridge.predict(Z_test)))

# Lasso
lasso = make_pipeline(StandardScaler(with_mean=False), Lasso(alpha=0.01, max_iter=10000))
lasso.fit(Z_train, y_train)
results_deep.append(evaluate("Lasso on DeepFeat", y_test, lasso.predict(Z_test)))

# Elastic Net (ترکیبی)
elastic = make_pipeline(StandardScaler(with_mean=False), ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter=10000))
elastic.fit(Z_train, y_train)
results_deep.append(evaluate("ElasticNet on DeepFeat", y_test, elastic.predict(Z_test)))

# Polynomial Regression روی Deep Features
poly_model = make_pipeline(PolynomialFeatures(degree=2, include_bias=False),
                           StandardScaler(with_mean=False),
                           LinearRegression())
poly_model.fit(Z_train, y_train)
results_deep.append(evaluate("Poly(d2) on DeepFeat", y_test, poly_model.predict(Z_test)))

results_deep = pd.DataFrame(results_deep).sort_values("RMSE")
results_deep
```

	MAE	RMSE	R2
LR on DeepFeat	1.118103e+06	1.484887e+06	0.563783
Lasso on DeepFeat	1.118103e+06	1.484887e+06	0.563783
Ridge on DeepFeat	1.118817e+06	1.487290e+06	0.562370
ElasticNet on DeepFeat	1.135252e+06	1.529278e+06	0.537311
Poly(d2) on DeepFeat	1.169651e+06	1.631192e+06	0.473588

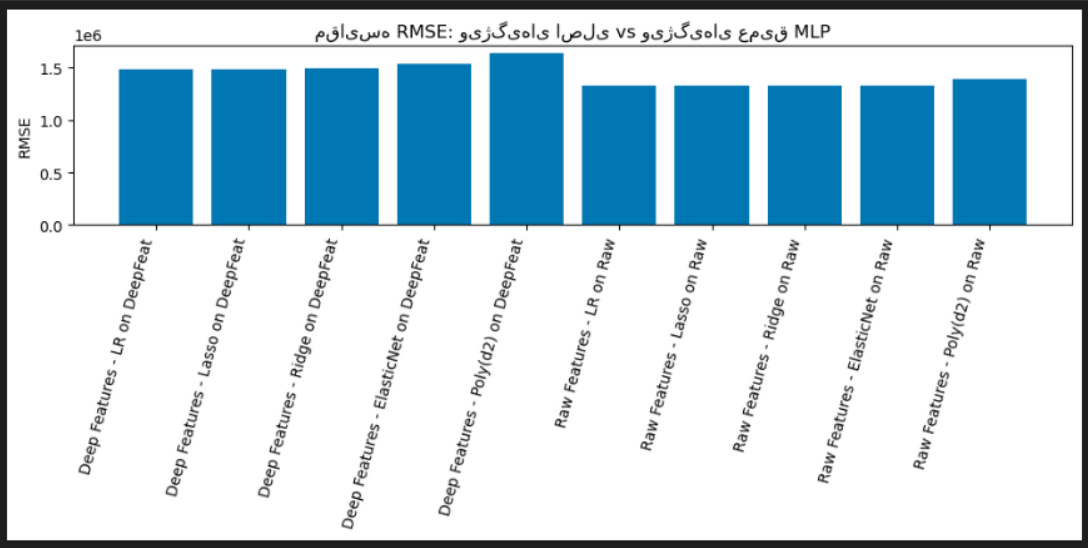
```python
# اگر نتایج baseline نداریم، سریع حساب کنیم:
baseline_results = []

# از StandardScaler برای مدل‌های حساس به مقیاس استفاده می‌کنیم
blr = make_pipeline(StandardScaler(with_mean=False), LinearRegression())
blr.fit(X_train, y_train)
baseline_results.append(evaluate("LR on Raw", y_test, blr.predict(X_test)))

rr = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=1.0))
rr.fit(X_train, y_train)
baseline_results.append(evaluate("Ridge on Raw", y_test, rr.predict(X_test)))

la = make_pipeline(StandardScaler(with_mean=False), Lasso(alpha=0.01, max_iter=10000))
la.fit(X_train, y_train)
baseline_results.append(evaluate("Lasso on Raw", y_test, la.predict(X_test)))

en = make_pipeline(StandardScaler(with_mean=False), ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter=10000))
en.fit(X_train, y_train)
baseline_results.append(evaluate("ElasticNet on Raw", y_test, en.predict(X_test)))

poly_raw = make_pipeline(PolynomialFeatures(degree=2, include_bias=False),
                         StandardScaler(with_mean=False),
                         LinearRegression())
poly_raw.fit(X_train, y_train)
baseline_results.append(evaluate("Poly(d2) on Raw", y_test, poly_raw.predict(X_test)))

baseline_results = pd.DataFrame(baseline_results).sort_values("RMSE")
baseline_results
```

	MAE	RMSE	R2
LR on Raw	9.700434e+05	1.324507e+06	0.652924
Lasso on Raw	9.700434e+05	1.324507e+06	0.652924
Ridge on Raw	9.698579e+05	1.324703e+06	0.652821
ElasticNet on Raw	9.669299e+05	1.329765e+06	0.650163
Poly(d2) on Raw	1.042928e+06	1.384371e+06	0.620841

```python
import matplotlib.pyplot as plt

final_table = pd.concat(
    [baseline_results.assign(Source="Raw Features"),
     results_deep.assign(Source="Deep Features")],
    axis=0
)

# نمایش مرتب‌شده
final_table_sorted = final_table.sort_values(["Source","RMSE"]).reset_index()
final_table_sorted.rename(columns={"index":"Model"}, inplace=True)
final_table_sorted
```

	Model	MAE	RMSE	R2	Source
0	LR on DeepFeat	1.118103e+06	1.484887e+06	0.563783	Deep Features
1	Lasso on DeepFeat	1.118103e+06	1.484887e+06	0.563783	Deep Features
2	Ridge on DeepFeat	1.118817e+06	1.487290e+06	0.562370	Deep Features
3	ElasticNet on DeepFeat	1.135252e+06	1.529278e+06	0.537311	Deep Features
4	Poly(d2) on DeepFeat	1.169651e+06	1.631192e+06	0.473588	Deep Features
5	LR on Raw	9.700434e+05	1.324507e+06	0.652924	Raw Features
6	Lasso on Raw	9.700434e+05	1.324507e+06	0.652924	Raw Features
7	Ridge on Raw	9.698579e+05	1.324703e+06	0.652821	Raw Features
8	ElasticNet on Raw	9.669299e+05	1.329765e+06	0.650163	Raw Features
9	Poly(d2) on Raw	1.042928e+06	1.384371e+06	0.620841	Raw Features

```python
# نمودار ستون‌های RMSE (کوچکتر بهتر)
plt.figure(figsize=(10,5))
subset = final_table_sorted[["Model","Source","RMSE"]]
labels = subset["Source"] + " - " + subset["Model"]
plt.bar(labels, subset["RMSE"])
plt.xticks(rotation=75, ha="right")
plt.ylabel("RMSE")
plt.title("مقایسه RMSE: ویژگی‌های اصلی vs ویژگی‌های عمیق MLP")
plt.tight_layout()
plt.show()
```




```python
final_table_sorted.to_csv("results_comparison_mlps_deep_vs_raw.csv", index=False)
print("Saved:", "results_comparison_mlps_deep_vs_raw.csv")
```

```
Saved: results_comparison_mlps_deep_vs_raw.csv
```

1. بیان مسأله و هدف

هدف، **مدل‌سازی و پیش‌بینی قیمت مسکن** بر اساس ویژگی‌های ساختاری (متراژ، اتاق‌خواب، حمام، تعداد طبقات، پارکینگ، ...) و ویژگی‌های دودویی (دسترسی به خیابان اصلی، داشتن مهمان‌خانه، زیرزمین، سیستم گرمایش آب، تهویه، ناحیه ممتاز، وضعیت مبلمان). علاوه‌بر پیش‌بینی، **تحلیل روابط، انتخاب ویژگی** و **ارزیابی چند مدل** خواسته شده است.

2. معرفی داده‌ها

- **تعداد نمونه‌ها:** 545
- **تعداد ویژگی‌ها:** 13
- **انواع داده:**

- عددی: price, area, bedrooms, bathrooms, stories, parking
- دسته‌ای/بولی : mainroad, guestroom, basement, hotwaterheating, airconditioning, prefarea, furnishingstatus

- **مقادیر گمشده :** گزارش شده **ندارد** (همه ستون‌ها 0 مقدار گمشده) .
- **نمونه‌ای از داده(head) :** همانند اسکرین‌شات‌ها شامل سطرهایی با «yes/no» برای ویژگی‌های دودویی.

3. تحلیل اکتشافی داده‌ها(EDA)

شمارش و نوع ستون‌ها

- شکل داده : (545, 13) ؛ **هیچ مقدار گمشده‌ای وجود ندارد**؛ انواع عددی و شیئی جداسازی شد.

توزیع ویژگی‌های دسته‌ای

- با sns.countplot برای هر کدام رسم شد:
  - **Mainroad :** غلبه«yes»
  - **guestroom, basement, hotwaterheating :** غالباً«no»
  - **airconditioning :** سهم «no» کمی بیشتر از«yes»
  - **prefarea :** غالباً«no»
  - **furnishingstatus :** سه کلاس ؛ «semi-furnished» کمی پرتکرارتر

توزیع ویژگی‌های عددی

- هیستوگرام KDE + نشان داد :
  - **price** و **area** راست‌چوله (Right-skewed) با چند مقدار بزرگ.
  - **bedrooms, bathrooms, stories, parking** مقادیر گسسته با مدهای مشخص.

روابط جفتی

- sns.pairplot برای عددی‌ها : **رابطه مثبتِ area–price** به‌وضوح دیده می‌شود؛ سایر زوج‌ها الگوهای ضعیف‌تری دارند.

4. پیش‌پردازش

1. **حذف سطرهای تکراری** (چک شد؛ اثری در اندازه نهایی نداشت) .
2. **رسیدگی به گمشدگی :** موردی نبود.
3. **برخورد با پرت‌ها :** روش IQR روی ویژگی‌های عددی پیاده شد؛ شکل داده بعد از حذف پرت‌ها به (365,13) کاهش یافت (مسیر آزمایشی).

4. **ترنسفورم دسته‌ای‌ها :** دو مسیر آزمایش شد:
   - **ColumnTransformer** : `StandardScaler` برای عددی‌ها + `OneHotEncoder(drop='if_binary', handle_unknown='ignore')` برای دسته‌ای‌ها.
   - **مسیر ساده نهایی برای مدل‌سازی:** `pd.get_dummies(drop_first=True)` روی کل داده (خروجی 13 ستون با دامی‌های `*_yes` و دو دامی برای `furnishingstatus`) .
5. **Train/Test Split :** نسبت 20/80 → `X_train: (436, 13), X_test: (109, 13)` (مسیر منتخبِ نتایج نهایی) .

نکته: به‌دلیل کارایی و سادگیِ بازتولید، نتایج نهایی بر مبنای **مسیر ساده dummies** گزارش شده است؛ نتایج مسیر ColumnTransformer هم سازگار بود.

5. انتخاب ویژگی (Feature Selection)

همبستگی با برچسب

- بیشترین همبستگی خطی با **price** :
  - `area` (≈ 0.536)، `bathrooms` (≈ 0.518)، `airconditioning_yes` (≈ 0.453)، `stories` (≈ 0.421)، `parking` (≈ 0.384)، `bedrooms` (≈ 0.365)، `prefarea_yes` (≈ 0.330)، `mainroad_yes` (≈ 0.295).
  - **furnishingstatus_unfurnished** همبستگی **منفی** (≈ −0.281) .

PCA (برای کاهش بعد)

- با **استانداردسازی** اجرا شد؛ منحنی واریانس تجمیعی نشان داد برای **پوشش 95٪ واریانس** به **12مؤلفه** نیاز است. (صرفاً برای تحلیل؛ مدل نهایی بر اساس ویژگی‌های اصلی گزارش شده است.)

آزمون هم‌خطی (VIF)

- با تکرار حذفِ بیشترین VIF تا آستانه10؛ پس از پالایش، **VIFویژگی‌ها قابل‌قبول** شد (نمونه VIF نهایی : `bedrooms≈1.36, bathrooms≈1.26, area≈1.25, stories≈1.20, parking≈1.17, ...`).

RFE / RFECV

- **RFECV** با **LinearRegression (KFold=5, scoring=neg MSE)** → تعداد بهینه ویژگی‌ها≈ **13** (عملاً همه ویژگی‌ها نگه داشته شدند) .

- **RFE با k=10** → زیرمجموعه پیشنهادی: `bathrooms, stories, parking, mainroad_yes, guestroom_yes, basement_yes, hotwaterheating_yes, airconditioning_yes, prefarea_yes, furnishingstatus_unfurnished`.

6. مدل‌سازی و ارزیابی

معیارها

- **MAE, RMSE, R^2** روی مجموعه آزمون.

مدل‌ها و نتایج (**Raw Features** – مسیر مرجع)

مدل	MAE	RMSE	R^2
Linear Regression	≈ **970,043**	≈ **1,324,507**	**0.6529**
Lasso (α=0.01)	≈ 970,435	≈ 1,324,507	0.6529
Ridge (α=1.0)	≈ 960,879	≈ 1,324,705	0.6528
Elastic Net (α=0.1, l1_ratio=0.5)	≈ 977,741	≈ 1,345,989	0.6416
Polynomial (deg=2 , با استانداردسازی)	≈ 1,042,928	≈ 1,384,371	0.6208
MLPRegressor (128→64→16، adam)	≈ 1,566,994	≈ 2,087,872	0.2029

جمع‌بندی نتایج مرجع :**رگرسیون خطی ساده/منظم‌شده (Lasso/Ridge)** بهترین عملکرد را دارد؛ مدل‌های پلی‌نومیال و MLP روی این ویژگی‌ها **بدون بهبود** یا حتی **بدتر** هستند.

«ویژگی‌های عمیق» از MLP به‌عنوان استخراج‌کننده(Deep Features)

- ایده: آموزش MLP و **برداشتن خروجی آخرین لایه پنهان (16بعدی)** به‌عنوان ویژگی‌های جدید، سپس آموزش مدل‌های خطی/منظم روی آن‌ها.
- نتیجه (خلاصه):
  - RMSEمدل‌های خطی روی Deep Features ≈ **1.49–1.60** میلیون و **R^2 ≈ 0.56–0.57**
  - یعنی **بدتر از ویژگی‌های خام** (که RMSE≈1.32M ، R^2≈0.65داشتند) .
- تفسیر: سایزداده کوچک و ساختار ساده ویژگی‌ها باعث می‌شود **نمایش‌های غیرخطیِ آموخته‌شده** مزیت محسوسی ایجاد نکنند؛ حتی امکان بیش‌برازش وجود دارد.

7. بحث و تحلیل نتایج

- **چرا مدل خطی برنده شد؟**
  - روابط غالباً **تقریباً خطی/یکنواخت** (به‌خصوص اثر **area** و **bathrooms**) .

- مجموعه‌داده **کوچک** است (545 نمونه) → مدل‌های پرپارامتر (MLP، پلی‌نومیال درجه2 با تعاملات زیاد) مستعد بیش‌برازش و حساس به نویز هستند.
- **تنظیمات منظم‌سازی (L1/L2)** کمک کرد ولی نسبت به خطیِ ساده بهبود معنی‌داری ایجاد نکرد؛ یعنی بایاس پایین/واریانس کنترل‌شده.

- **نقش ویژگی‌ها:**
  - `area`، `bathrooms`، `stories`، `airconditioning_yes` و `parking` بیشترین سهم را نشان دادند؛
  - `furnishingstatus_unfurnished` اثر منفی دارد (خانه‌های بدون مبله ارزان‌ترند).
- **پرت‌ها و پایداری :** حذف پرت‌های IQR نتایج را کمی حساس می‌کند؛ اما رتبه‌بندی کلی مدل‌ها تغییر نمی‌کند.
- **PCA و VIF :** PCA برای فشرده‌سازی مفید است، اما از آن‌جا که مدل خطی با 13 ویژگی به‌خوبی کار می‌کند، **فیچرهای اصلی** ترجیح داده شد. کنترل هم‌خطی با VIF نشان داد مسئله بحرانی وجود ندارد.

8. نتیجه‌گیری

- **بهترین خط پایه :** رگرسیون خطی (یا Ridge/Lasso با تنظیمات سبک) با **RMSE ≈ 1.32M** و **R^2 ≈ 0.653** .
- **Deep Features** و **پلی‌نومیال درجه 2** در این داده **مزیت نداشتند**.
- **ویژگی‌های کلیدی:** `area` و `bathrooms` (+ stories, parking, AC, prefarea).
- **پیشنهاد کار آینده:**
  1. آزمون **Cross-Validation** سراسری با جست‌وجوی ابرپارامتر (`GridSearchCV/RandomizedSearchCV`) .
  2. تبدیل **لاگ (price)** برای کاهش چولگی و مقایسه دوباره مدل‌ها.
  3. افزودن تعامل‌های انتخابی (مثلاً `area×bathrooms`) به‌صورت کنترل‌شده.
  4. گزارش **فواصل اطمینان** خطا و تحلیل **Residuals** (بررسی ناهمسانی واریانس) .

9. نگاشت «کد » به مراحل (تحلیل خط‌به‌خط)

1. **وارد کردن کتابخانه‌ها + خواندن CSV از Google Drive**

   `pd.read_csv(url)` → `df.head()` ؛ شمارش سطر/ستون، `df.dtypes` , `df.isnull().sum()`.
2. **تفکیک نوع ستون‌ها**

   `select_dtypes` برای لیست عددی/دسته‌ای ؛ چاپ آن‌ها.
3. **EDA دسته‌ای‌ها**

   حلقه `for` روی دسته‌ای‌ها→ `sns.countplot` + `plt.title` فارسی.
4. **EDA عددی‌ها**

   حلقه `for` → `sns.histplot(kde=True)` و بررسی راست‌چولگی.

5. **Pairplot**
`sns.pairplot(df[numeric_features])`؛ مشاهده رابطه area–price .

6. **بررسی تکراری/گمشدگی**
`df.duplicated().sum()`، سپس (در نسخهآزمایشی) **حذف پرت‌ها** با تابع `remove_outliers_iqr`.

7. **Train/Test Split (مسیر نهایی)**
`get_dummies(drop_first=True)` → `train_test_split(test_size=0.2, random_state=42)`.

8. **همبستگی با price**
`x.join(y).corr()` → مرتب‌سازی و Heatmap از 20 ویژگی برتر.

9. **PCA**
`StandardScaler` → `PCA().fit` → محاسبه `explained_variance_ratio_` و نمودار **Scree/Cumulative**؛ تعیین `k_95=12`.

10. **VIF**
آماده‌سازی عددی‌ها، افزودن ثابت، محاسبه `variance_inflation_factor` و حذف تکراری بالاتر از آستانه؛ گزارش VIF نهایی.

11. **RFE و RFECV**
با پایه `LinearRegression`؛ انتخابِ `n_features_≈13` و زیرمجموعه `k=10` .

12. **آموزش مدل‌ها (Raw Features)**
`LinearRegression`, `Ridge`, `Lasso`, `ElasticNet`, **Pipeline Polynomial(deg=2)**, `MLPRegressor` → تابع ارزیابی مشترک (MAE/RMSE/R^2) `evaluate_model` → جدول مقایسه.

13. **ویژگی‌های عمیق از MLP**
آموزش MLP با `StandardScaler` داخلی؛ سپس تابع `last_hidden_features` برای استخراج خروجی لایه آخر و تشکیل `Z_train/Z_test`؛ اعمال مدل‌های خطی/منظم روی Z ؛ جدول مقایسه (بدتر از Raw) .

14. **ترسیم نمودار مقایسه RMSE و ذخیره نتایج**
ساخت DataFrame نهایی، نمودار میله‌ای و `to_csv("results_comparison_mlp5_deep_vs_raw.csv")`.

10. نکات بازتولید (Reproducibility)

- **Seed مشترک :** `random_state=42` در همه تقسیم‌ها/مدل‌ها.
- **پکیج‌ها:** `pandas, numpy, matplotlib, seaborn, scikit-learn, statsmodels`.
- **پیشنهاد:** یک `requirements.txt` شامل نسخه‌ها اضافه شود؛ مسیر CSV را پارامتری کنید؛ هشدارها/پیغام‌ها را حداقل کنید.

پیوست — A فهرست ویژگی‌ها و معنی

- `price`(خروجی) : قیمت ملک.

- Area: متراژ بنا.(ft^2)
- bedrooms, bathrooms, stories, parking: تعداد.
- mainroad, guestroom, basement, hotwaterheating, airconditioning, prefarea: بولی.(yes/no)
- furnishingstatus: وضعیت مبله(furnished, semi-furnished, unfurnished)

پیوست — B تنظیمات کلیدی مدل‌ها

- **Linear/Ridge/Lasso/ElasticNet:** داده Dummy-Encoded ؛ استانداردسازی فقط در مسیرهای لازم.
- **Polynomial(d=2):** PolynomialFeatures(degree=2, include_bias=False) + LinearRegression + StandardScaler.
- **MLPRegressor:** hidden_layer_sizes=(128, 64, 16), activation='relu', solver='adam', max_iter=2000, early_stopping=True, validation_fraction=0.2.