دوره جامع علم داده دانشگاه تهران

پروژه تحلیل داده با پایتون

بررسی سکته قلبی در بین افراد یک جامعه

استاد : دکتر امیررضا تجلی

مجری : رضا عبدالوهابی

پاییز ۱۴۰۳

توضیح کلی در مورد دیتاست

این دیتاست شامل اطلاعات مربوط به افرادی است که دچار سکته قلبی شده‌اند. هدف از جمع‌آوری این داده‌ها، تحلیل عوامل مؤثر بر بروز سکته قلبی و شناسایی ویژگی‌های مرتبط با این بیماری است. هر سطر در دیتاست نمایانگر یک فرد است و ستون‌ها شامل ویژگی‌های مختلفی هستند که می‌توانند به پیش‌بینی خطر سکته قلبی کمک کنند.

معرفی ستون‌های دیتاست

۱) Age (سن):

این ستون نشان‌دهنده سن افراد به سال است. سن یکی از عوامل خطر مهم در بروز بیماری‌های قلبی است و معمولاً با افزایش سن، خطر سکته قلبی نیز افزایش می‌یابد.

۲) Sex (جنسیت):

این ستون نشان‌دهنده جنسیت فرد است که معمولاً به صورت ۰ (زن) و ۱ (مرد) کدگذاری می‌شود. جنسیت می‌تواند تأثیر زیادی بر روی ریسک بیماری‌های قلبی داشته باشد.

۳) Chest Pain (درد قفسه سینه):

این ستون اطلاعاتی در مورد نوع درد قفسه سینه فرد ارائه می‌دهد. معمولاً این نوع درد به چند دسته تقسیم می‌شود، که می‌تواند شامل درد غیرقابل توجیه، درد ناشی از فعالیت، درد ناشی از استراحت و غیره باشد. نوع درد قفسه سینه می‌تواند نشان‌دهنده مشکلات قلبی باشد.

۴) Blood Pressure (فشار خون):

این ستون نشان‌دهنده فشار خون سیستولیک و دیاستولیک فرد است، که به میلی‌متر جیوه (mmHg) اندازه‌گیری می‌شود. فشار خون بالا یکی از عوامل خطر شناخته‌شده برای بیماری‌های قلبی است.

۵) Cholesterol (کلسترول):

این ستون سطح کلسترول خون فرد را نشان می‌دهد. کلسترول بالا می‌تواند به انسداد عروق و در نتیجه به سکته قلبی منجر شود.

۶) Blood Sugar (قند خون):

این ستون نشان‌دهنده سطح قند خون فرد است و معمولاً به صورت ۰ (کمتر از ۱۲۰ mg/dl) و ۱ (بیشتر یا مساوی ۱۲۰ mg/dl) کدگذاری می‌شود. قند خون بالا می‌تواند به دیابت و بیماری‌های قلبی مرتبط باشد.

۷) Electrocardiographic (الکتروکاردیوگرام):

این ستون اطلاعاتی در مورد نتایج الکتروکاردیوگرام (ECG) فرد ارائه می‌دهد. این نتایج می‌تواند نشان‌دهنده وجود مشکلات قلبی مانند آریتمی یا ایسکمی باشد.

۸) Heart Rate (ضربان قلب):

این ستون نشان‌دهنده تعداد ضربان قلب فرد در دقیقه است. ضربان قلب بالا یا پایین می‌تواند به مشکلات قلبی اشاره کند.

۹) Exercise Induced (تحریک ناشی از ورزش):

این ستون نشان‌دهنده واکنش قلب به ورزش است و معمولاً به صورت ۰ (بدون تحریک) و ۱ (تحریک) کدگذاری می‌شود. این اطلاعات می‌تواند به ارزیابی سلامت قلب کمک کند.

۱۰) Depression (افسردگی):

این ستون نشان‌دهنده سطح افسردگی فرد است که می‌تواند به صورت عددی یا کدگذاری شده باشد. افسردگی یکی از عوامل خطر برای بیماری‌های قلبی است.

۱۱) Slope (شیب):

این ستون نشان‌دهنده شیب منحنی استراحت در تست ورزش است. شیب می‌تواند به ارزیابی وضعیت قلب و عروق کمک کند.

۱۲) Ca (کلسیم):

این ستون نشان‌دهنده تعداد عروق مسدود شده است که به صورت عددی کدگذاری می‌شود. تعداد عروق مسدود شده می‌تواند به شدت بیماری قلبی اشاره کند.

۱۳) Thal (ثالاسمی):

این ستون نشان‌دهنده وضعیت ثالاسمی فرد است و معمولاً به صورت ۰ (بدون ثالاسمی)، ۱ (ثالثی) و ۲ (ثالثی معکوس) کدگذاری می‌شود. وضعیت ثالاسمی می‌تواند تأثیر زیادی بر روی سلامت قلب داشته باشد.

۱۴) C (بیماری قلبی):

این ستون نشان‌دهنده وجود یا عدم وجود بیماری قلبی در فرد است که معمولاً به صورت ۰ (بدون بیماری) و ۱ (با بیماری) کدگذاری می‌شود. این ستون هدف اصلی تحلیل و پیش‌بینی در این دیتاست است.

این دیتاست می‌تواند به تحلیل و شناسایی الگوهای مرتبط با سکته قلبی کمک کند و به محققان و پزشکان در پیشگیری و درمان این بیماری یاری رساند. با بررسی دقیق هر یک از این ستون‌ها، می‌توان عوامل خطر را شناسایی و راهکارهای مؤثری برای کاهش خطر سکته قلبی ارائه داد.

در این پروژه با بررسی عوامل موثر و تجزیه و تحلیل اطلاعات مدل هایی را اجرا می کنیم تا بهترین دقت در تعیین افراد جدیدی با اطلاعات مشابه را براساس اینکه دچار سکته قلبی می شوند یا خیر را بدست بیاوریم.

در ادامه توضیح هر بخش به طور کامل ارائه خواهد شد.

مرحله ۱: بارگذاری کتابخانه‌ها و داده‌ها

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

در این مرحله، ما ابتدا کتابخانه‌های مورد نیاز را بارگذاری می‌کنیم.

کتابخانه Pandas:

Pandas یک کتابخانه بسیار قدرتمند برای تحلیل داده‌ها در پایتون است. این کتابخانه به ویژه برای کار با داده‌های جدولی (DataFrame) طراحی شده است.

با استفاده از Pandas می‌توانیم داده‌ها را بارگذاری، تمیز، تحلیل و پردازش کنید. این کتابخانه شامل ابزارهایی برای خواندن و نوشتن داده‌ها از / به فرمت‌های مختلف (مانند CSV، Excel و SQL) و همچنین توابعی برای انجام عملیات‌های مختلف روی داده‌ها (مانند گروه‌بندی، ادغام و فیلتر کردن) است.

کتابخانه NumPy:

NumPy یک کتابخانه پایه برای محاسبات عددی در پایتون است. این کتابخانه به ویژه برای کار با آرایه‌های چند بعدی (ndarray) و انجام عملیات‌های ریاضی و علمی طراحی شده است.

NumPy به ما اجازه می‌دهد که محاسبات ریاضی پیچیده را به راحتی انجام دهید، به ویژه بر روی داده‌های عددی. این کتابخانه شامل توابعی برای انجام عملیات‌های ریاضی، جبر خطی، و تولید اعداد تصادفی است.

کتابخانه Matplotlib:

Matplotlib یک کتابخانه قدرتمند برای ترسیم و تجسم داده‌ها در پایتون است. زیرمجموعه pyplot این کتابخانه، به ما امکان می‌دهد تا به راحتی نمودارها و گراف‌ها را ایجاد کنیم.

با استفاده از Matplotlib، می‌توان انواع مختلفی از نمودارها (مانند خطی، میله‌ای، پراکندگی و هیستوگرام) را رسم کرد. این کتابخانه به ویژه در تحلیل داده‌ها و ارائه نتایج بصری بسیار مفید است.

کتابخانه Seaborn:

Seaborn یک کتابخانه تجسم داده‌ها است که بر پایه Matplotlib ساخته شده است. این کتابخانه به ما امکان می‌دهد تا نمودارهای زیبا و پیچیده‌تری را با کد کمتر ایجاد کنیم.

Seaborn شامل توابعی برای ایجاد نمودارهای آماری، مانند نمودارهای توزیع، جعبه‌ای و حرارتی است. این کتابخانه به ویژه برای تحلیل داده‌های آماری و بصری‌سازی روابط بین متغیرها بسیار مفید است.

بارگذاری داده‌ها

```
# آدرس دهی دیتاست
file_path = 'C:/Users/IRS/Heart data.csv'

# فراخوانی دیتاست
df = pd.read_csv(file_path)

# نمایش چند سطر ابتدایی داده ها #
df.head()
```

در این خط، ما یک متغیر به نام file_path تعریف می‌کنیم که شامل مسیر فایل CSV است. این مسیر مشخص می‌کند که فایل داده‌ها در کجا قرار دارد. مطمئن می‌شویم که این آدرس صحیح است و فایل در آن مکان وجود دارد.

فراخوانی دیتاست

در این مرحله، ما از تابع read_csv کتابخانه Pandas استفاده می‌کنیم تا داده‌ها را از فایل CSV بخوانیم. این تابع فایل را بارگذاری کرده و داده‌ها را به یک DataFrame تبدیل می‌کند. متغیر df حالا شامل داده‌های ما است و می‌توانیم با آن کار کنیم.

نمایش چند سطر ابتدایی داده‌ها

با استفاده از تابع ()head ، ما می‌توانیم پنج سطر اول DataFrame را مشاهده کنیم. این کار به ما کمک می‌کند تا نگاهی سریع به ساختار و محتوای داده‌ها بیندازیم و بررسی کنیم که آیا داده‌ها به درستی بارگذاری شده‌اند یا خیر.

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
0	63	1	1	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	37	1	3	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
2	41	0	2	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
3	56	1	2	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0
4	57	0	4	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0

برای اینکه تعداد سطر و ستون ها را مشاهده کنیم از دستور df.shape استفاده کنیم.

توضیح: این تاپل شامل دو عدد است: عدد اول: تعداد سطرها (rows) عدد دوم: تعداد ستون ها(columns)

```
1 df.shape
```

```
(597, 14)
```

دستور df.columns در کتابخانه ی Pandas در پایتون برای نمایش نام های ستون های یک DataFrame استفاده می شود. این دستور یک Index از نام های ستون های DataFrame را برمی گرداند و به ما کمک می کند تا به نام ستون ها دسترسی پیدا کنیم و آن ها را بررسی کنیم. ممکن است در نام ستون ها خطای املایی یا کاراکتر اضافه ای باشد.

```
1 df.columns
```

```
Index(['Age (age in year)', 'sex', 'chest pain', 'blood pressure',
       'cholestoral ', 'blood sugar', 'electrocardiographic ', 'heart rate',
       'exercise induced', 'depression ', 'slope', 'ca', 'thal', 'c'],
      dtype='object')
```

بررسی داده ها

در این مرحله، تعداد کل سطرها و ستون ها و همچنین نوع داده های هر ستون را بررسی می کنیم. این دستور نشان می دهد که چند ستون و سطر داریم و نوع داده های هر ستون را مشخص می کند. همچنین نشان می دهد که آیا داده های مفقود داریم یا خیر.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 597 entries, 0 to 596
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age (age in year)      597 non-null    int64
 1   sex                    597 non-null    int64
 2   chest pain             597 non-null    int64
 3   blood pressure         596 non-null    float64
 4   cholestoral            574 non-null    float64
 5   blood sugar            589 non-null    float64
 6   electrocardiographic   596 non-null    float64
 7   heart rate             596 non-null    float64
 8   exercise induced       596 non-null    float64
 9   depression             597 non-null    float64
 10  slope                  407 non-null    float64
 11  ca                     303 non-null    float64
 12  thal                   329 non-null    float64
 13  c                      597 non-null    int64
dtypes: float64(10), int64(4)
memory usage: 65.4 KB
```

دستور pd.set_option در کتابخانه ی Pandas به ما این امکان را می دهد که تنظیمات مربوط به نمایش DataFrame ها را تغییر دهیم.

نکته: اگر DataFrame بسیار بزرگ باشد، نمایش تمام سطرها و ستون ها ممکن است باعث ایجاد خروجی بسیار طولانی شود و خواندن آن دشوار باشد. در چنین مواردی، می توانیم از تنظیمات پیش فرض استفاده کنیم یا فقط بخشی از DataFrame را با استفاده از df.head یا df.tail مشاهده کنیم.

```
1 pd.set_option("display.max_rows", None, "display.max_columns", None)
2 df
```

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
0	63	1	1	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	37	1	3	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
2	41	0	2	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
3	56	1	2	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0
4	57	0	4	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0
5	57	1	4	140.0	192.0	0.0	0.0	148.0	0.0	0.4	2.0	0.0	6.0	0

بررسی رابطه سن و جنسیت با فشارخون بر اساس تعداد افرادی که سکته کرده اند یا خیر. در ستون اول جنسیت بر اساس میانگین سن افرادی که دچار سکته شده اند یا بلعکس نمایش داده شده است و ستون سوم حداکثر مقدار فشار خون این افراد آمده و در ستون c تعداد هر گروه مشخص شده است.

```
q = df.groupby(['c' , 'sex']).agg({'Age (age in year)' : np.mean, 'blood pressure': np.max, 'c':np.size})
q
```

c	sex	Age (age in year)	blood pressure	c
0	0	51.035461	180.0	141
	1	48.549763	190.0	211
1	0	55.918919	200.0	37
	1	53.110577	200.0	208

مرحله ۲: تحلیل اکتشافی داده‌ها (EDA)

تحلیل اکتشافی به ما کمک می کند که یک تصویر کلی از داده‌ها داشته باشیم. میتوان از کتابخانه های موجود در پایتون مثل summarytool یا از پکیج های پیشرفته تر استفاده کرد. در اینجا از کتابخانه dtale استفاده کردیم.

```
# !pip install Dtale

import dtale
dtale.show(df)
```

D-TALE Actions Visualize Highlight Settings

▸ 597	14	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced
0		63	1	1	145.00	233.00	1.00	2.00	150.00	0.00
1		37	1	3	130.00	250.00	0.00	0.00	187.00	0.00
2		41	0	2	130.00	204.00	0.00	2.00	172.00	0.00
3		56	1	2	120.00	236.00	0.00	0.00	178.00	0.00
4		57	0	4	120.00	354.00	0.00	0.00	163.00	1.00
5		57	1	4	140.00	192.00	0.00	0.00	148.00	0.00
6		56	0	2	140.00	294.00	0.00	2.00	153.00	0.00
7		44	1	2	120.00	263.00	0.00	0.00	173.00	0.00
8		52	1	3	172.00	199.00	1.00	0.00	162.00	0.00
9		57	1	3	150.00	168.00	0.00	0.00	174.00	0.00
10		54	1	4	140.00	239.00	0.00	0.00	160.00	0.00
11		48	0	3	130.00	275.00	0.00	0.00	139.00	0.00
12		49	1	2	130.00	266.00	0.00	0.00	171.00	0.00
13		64	1	1	110.00	211.00	0.00	2.00	144.00	1.00

کتابخانه Dtale

کتابخانه Dtale یک ابزار بسیار مفید برای تجزیه و تحلیل و تجسم داده‌ها در پایتون است. این کتابخانه به ما این امکان را می‌دهد که به صورت تعاملی با داده‌ها کار کنیم و به سرعت بینش‌های مختلف را از آن‌ها استخراج کنیم.

ویژگی‌های کلیدی Dtale

✓ تجزیه و تحلیل تعاملی:

Dtale به ما اجازه می‌دهد تا داده‌ها را به صورت تعاملی مشاهده کنیم. ما می‌توانیم فیلترها، مرتب‌سازی‌ها و جستجوهای مختلفی را بر روی داده‌ها انجام دهیم و تغییرات را به صورت آنی مشاهده کنیم.

✓ رابط کاربری وب:

پس از فراخوانی dtale.show(df), یک رابط کاربری وب باز می‌شود که به ما امکان می‌دهد داده‌ها را در یک محیط گرافیکی مشاهده کنیم. این رابط به ما این امکان را می‌دهد که بدون نیاز به نوشتن کد اضافی، تجزیه و تحلیل‌های مختلفی را انجام دهیم.

✓ امکانات تجسم داده‌ها:

Dtale امکاناتی برای تجسم داده‌ها فراهم می‌کند، از جمله نمودارها و گراف‌ها. این ویژگی به ما کمک می‌کند تا الگوها و روابط موجود در داده‌ها را بهتر درک کنیم.

✓ پشتیبانی از انواع مختلف داده‌ها:

این کتابخانه می‌تواند با انواع مختلف داده‌ها کار کند، از جمله DataFrame های Pandas، که این امر آن را برای تحلیل داده‌ها در علم داده و یادگیری ماشین بسیار مفید می‌سازد.

✓ امکان ذخیره و بارگذاری تنظیمات:

Dtale به ما این امکان را می‌دهد که تنظیمات و فیلترهای خود را ذخیره کنیم و در آینده دوباره بارگذاری کنیم. این ویژگی می‌تواند در تحلیل‌های مکرر بسیار مفید باشد.

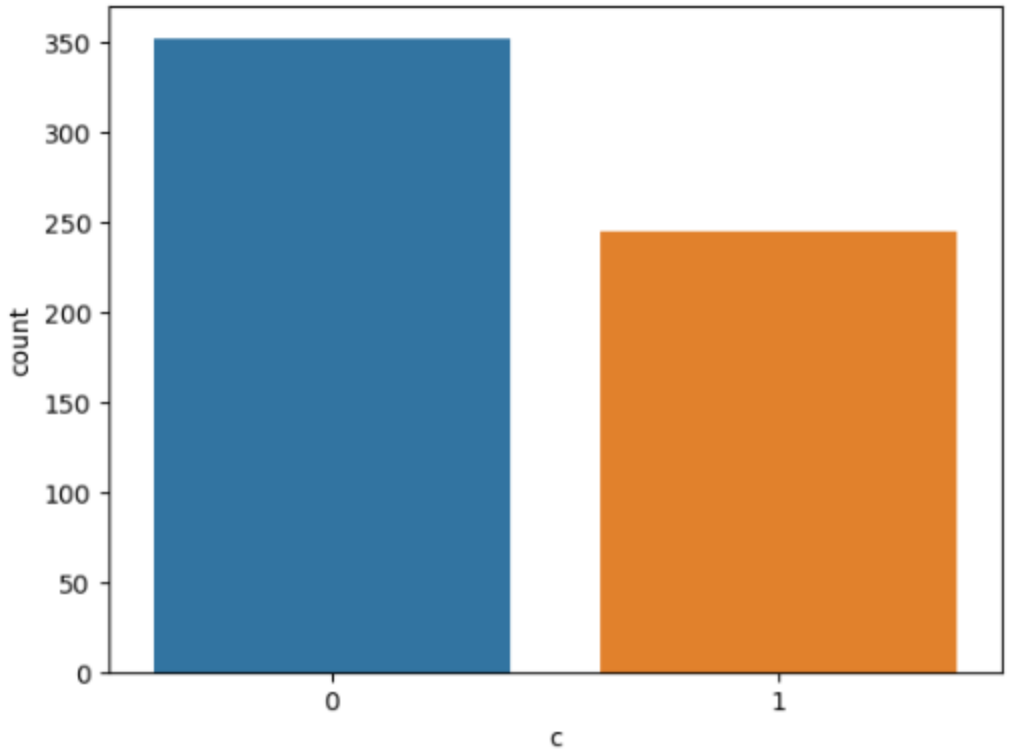
بررسی وضعیت ستون تارگت

با استفاده از دستور value_counts میتوانیم تعداد افرادی که سکته کرده اند یا بلعکس را مشاهده کنیم و برای درک بهتر روی نموار countplot پیاده سازی کنیم.

```
print(df.c.value_counts())
sns.countplot(x='c', data=df)
plt.show()
```

خروجی ↓

```
c
0    352
1    245
Name: count, dtype: int64
```



تحلیل آماری

به ما کمک می کند که یک تصویر کلی از داده ها داشته باشیم. برای مثال، با استفاده از دستور df.describe می توانیم مقادیر آماری مانند میانگین، میانه و انحراف معیار ، حداقل، حداکثر و ... ستون های عددی را مشاهده کنیم.

توضیح: این اطلاعات به ما کمک می کند تا محدوده مقادیر هر ویژگی را بشناسیم.

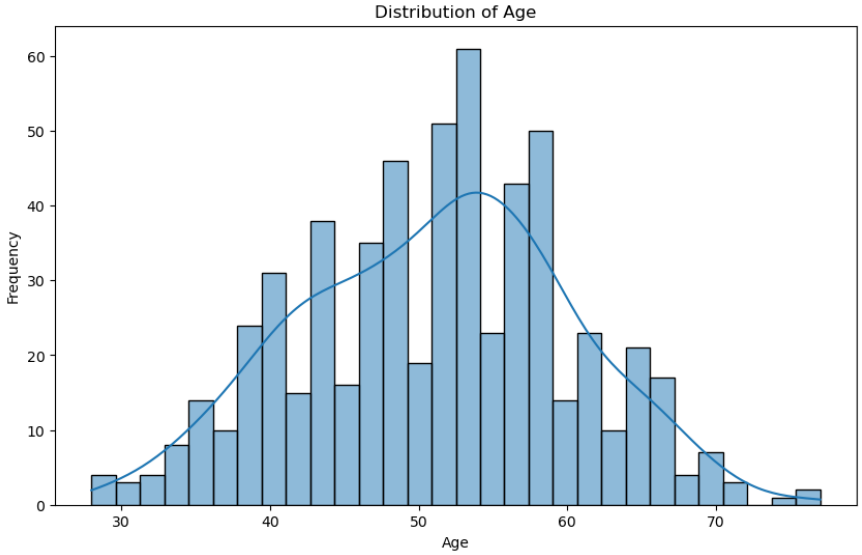
```
# نمایش توزیع آماری داده ها تا 4 رقم اعشار
df.describe().applymap(lambda x: f"{x:.4f}" if isinstance(x, float) else x)
```

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	
count	597.0000	597.0000	597.0000	596.0000	574.0000	589.0000	596.0000	596.0000	596.0000	597.0000	407.0000	303.0000	329.0000	597.00
mean	51.1826	0.7018	3.0720	132.1292	248.6551	0.1104	0.6107	144.4564	0.3154	0.8162	1.6757	0.6931	4.8116	0.41
std	9.0744	0.4578	0.9658	17.6038	59.7848	0.3136	0.8694	23.7943	0.4651	1.0679	0.5728	1.0492	1.9289	0.49
min	28.0000	0.0000	1.0000	92.0000	85.0000	0.0000	0.0000	71.0000	0.0000	0.0000	1.0000	0.0000	3.0000	0.00
25%	44.0000	0.0000	2.0000	120.0000	211.0000	0.0000	0.0000	128.0000	0.0000	0.0000	1.0000	0.0000	3.0000	0.00
50%	52.0000	1.0000	3.0000	130.0000	242.5000	0.0000	0.0000	146.0000	0.0000	0.2000	2.0000	0.0000	3.0000	0.00
75%	58.0000	1.0000	4.0000	140.0000	278.7500	0.0000	2.0000	162.0000	1.0000	1.5000	2.0000	1.0000	7.0000	1.00
max	77.0000	1.0000	4.0000	200.0000	603.0000	1.0000	2.0000	202.0000	1.0000	6.2000	3.0000	9.0000	7.0000	1.00

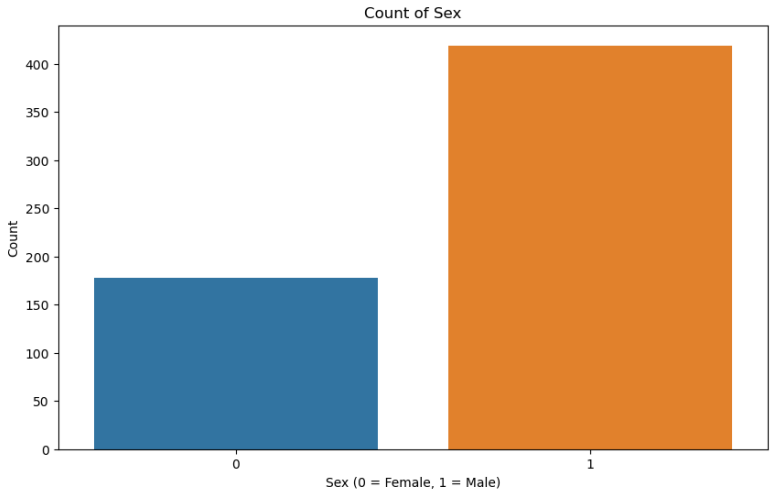
در این مرحله، ما ادامه تحلیل اکتشافی داده ها (EDA) را انجام می دهیم. این شامل بررسی توزیع ویژگی های مختلف و شناسایی الگوهای موجود در داده ها است. به عنوان مثال:

توزیع سن: توزیع سن افراد را بررسی می‌کنیم تا ببینیم آیا سن در بروز سکته قلبی تأثیر دارد یا خیر. توزیع جنسیت: تعداد مردان و زنان در داده‌ها را بررسی می‌کنیم. توزیع درد قفسه سینه: انواع مختلف درد قفسه سینه را بررسی می‌کنیم. همبستگی: با استفاده از ماتریس همبستگی، ارتباط میان ویژگی‌ها و متغیر هدف را بررسی می‌کنیم.

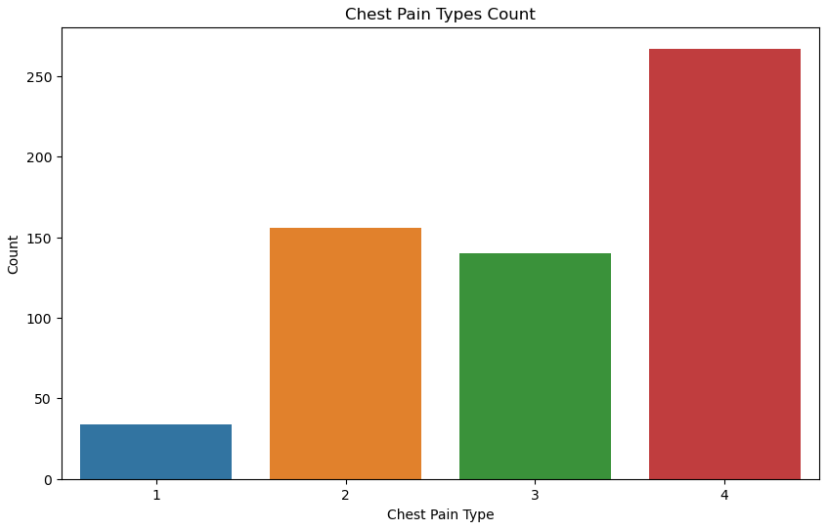
```python
# 1. توزیع سن
plt.figure(figsize=(10, 6))
sns.histplot(df['Age (age in year)'], bins=30, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```




```python
# 2. توزیع جنسیت
plt.figure(figsize=(10, 6))
sns.countplot(x='sex', data=df)
plt.title('Count of Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.ylabel('Count')
plt.show()
```



```python
# 3. توزیع درد قفسه سینه
plt.figure(figsize=(10, 6))
sns.countplot(x='chest pain', data=df)
plt.title('Chest Pain Types Count')
plt.xlabel('Chest Pain Type')
plt.ylabel('Count')
plt.show()
```



ماتریس همبستگی

```python
# همبستگی میان ویژگی ها
plt.figure(figsize=(12, 8))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

✓ تنظیم اندازه تصویر

این خط از تابع figure() کتابخانه Matplotlib استفاده می‌کند تا اندازه تصویر را تنظیم کند.

با تعیین figsize=(۱۲, ۸) , ما اندازه تصویر را به ۱۲ اینچ عرض و ۸ اینچ ارتفاع تنظیم می‌کنیم. این کار به ما کمک می‌کند تا نقشه حرارتی به وضوح قابل مشاهده باشد و جزئیات آن بهتر دیده شود.

✓ محاسبه ماتریس همبستگی

در این خط، ما از تابع corr() کتابخانه Pandas استفاده می‌کنیم تا ماتریس همبستگی بین ویژگی‌های مختلف df را محاسبه کنیم. ماتریس همبستگی یک جدول است که نشان‌دهنده همبستگی بین هر جفت از ویژگی‌ها است. مقادیر همبستگی می‌توانند بین -۱ و ۱ متغیر باشند:

- ۱: همبستگی مثبت کامل (با افزایش یکی، دیگری نیز افزایش می‌یابد).

- -۱: همبستگی منفی کامل (با افزایش یکی، دیگری کاهش می‌یابد).
- ۰: عدم همبستگی (هیچ رابطه‌ای وجود ندارد).

✓ ایجاد نقشه حرارتی

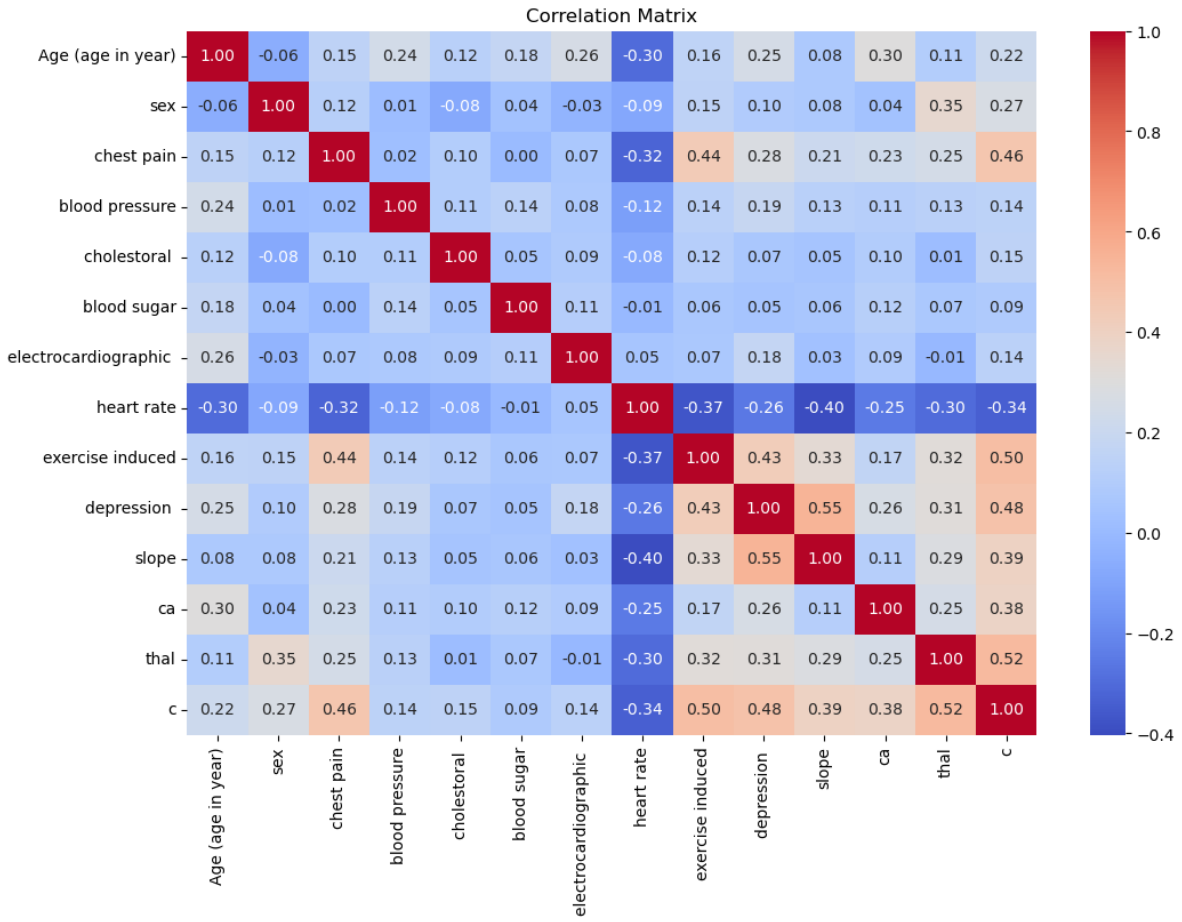
در این خط، از تابع heatmap() کتابخانه Seaborn برای ایجاد نقشه حرارتی استفاده می‌شود.

کاربرد:

- corr_matrix: ماتریس همبستگی که قبلاً محاسبه شده است.
- annot=True: این گزینه مشخص می‌کند که مقادیر همبستگی درون هر خانه نقشه حرارتی نمایش داده شوند.
- cmap='coolwarm': این گزینه رنگ‌بندی نقشه حرارتی را تعیین می‌کند. در اینجا، از رنگ‌های سرد و گرم برای نشان دادن مقادیر مثبت و منفی استفاده می‌شود.
- fmt='.2f': این گزینه فرمت نمایش مقادیر را تعیین می‌کند، به طوری که مقادیر با دو رقم اعشار نمایش داده شوند.

✓ عنوان‌دهی و نمایش تصویر

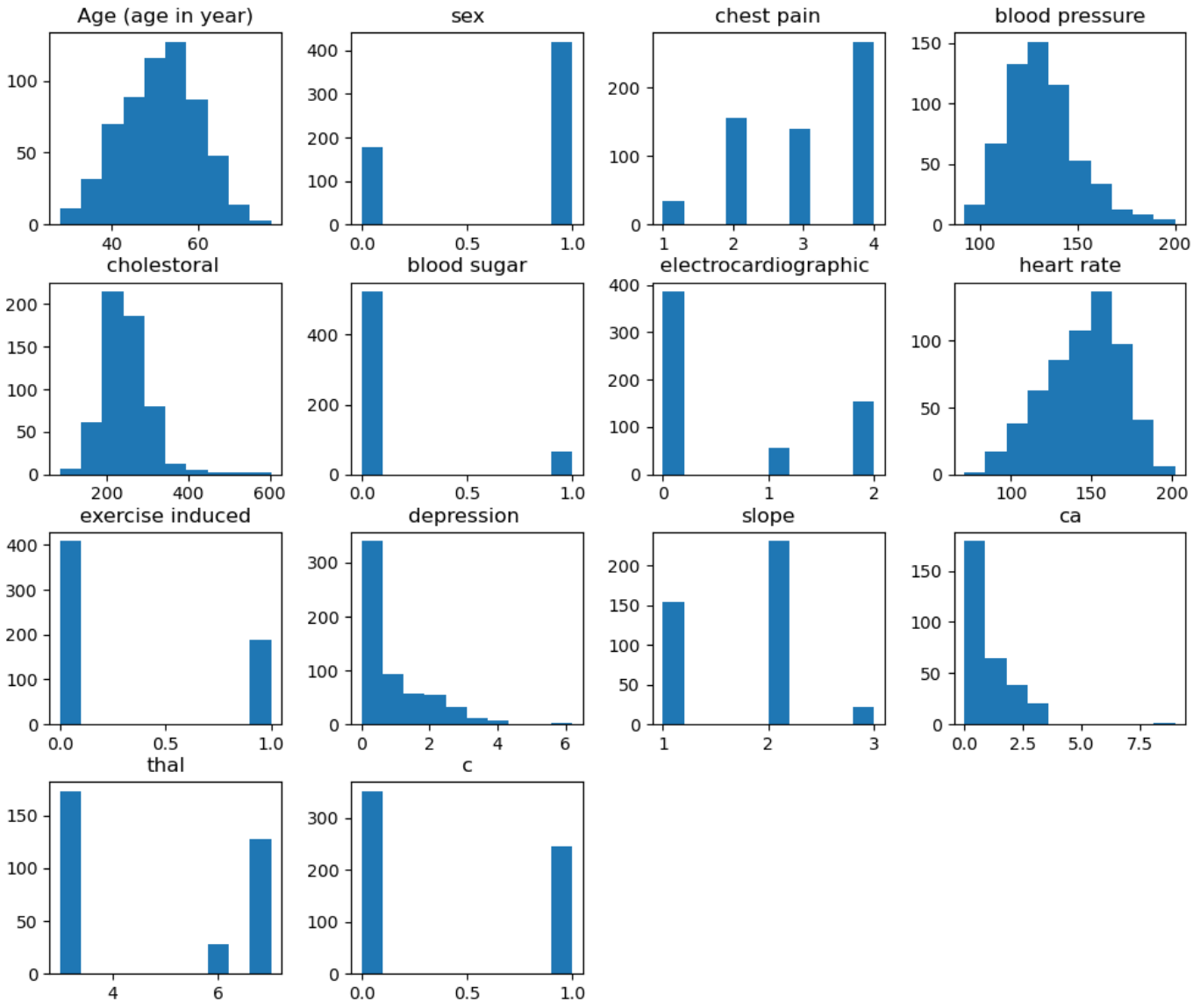
در این دو خط، ابتدا عنوان نقشه حرارتی را با استفاده از title() تنظیم می‌کنیم و سپس با استفاده از show() تصویر را نمایش می‌دهیم. این کار به ما کمک می‌کند تا نتیجه را به صورت واضح و با عنوان مناسب مشاهده کنیم.



مشاهده نمودار هیستوگرام داده ها به صورت یکجا

برای درک بهتر وضعیت داده های موجود در ستون ها با استفاده از نمودار هیستوگرام میتوانیم توزیع هر ستون را مشاهده کنیم.

```
# Visualize data distributions
df.hist(figsize=(12, 10), grid = False)
plt.show()
```



مدیریت داده های مفقودهMissing Data

در این مرحله، ما به بررسی داده های مفقود می پردازیم. ابتدا تعداد داده های مفقود در هر ستون (NaN) را شناسایی می کنیم. سپس برای جایگزینی داده های مفقود: برای ویژگی های کیفی (داده های دسته ای)، از مد (mode) و برای ویژگی های کمی، از میانه (median) استفاده میکنیم. این کار به ما کمک میکند تا از حذف دادهها جلوگیری کنیم و اطلاعات

بیشتری را حفظ کنیم. این کد تعداد داده های مفقود در هر ستون را نمایش می دهد تا بدانیم چه تعداد داده در هر ستون ناقص است.

```
# شناسایی و نمایش تعداد داده های مفقود در هر ستون
print("Missing values in each feature:\n", df.isnull().sum())
```

```
Missing values in each feature:
 Age (age in year)          0
sex                         0
chest pain                  0
blood pressure              1
cholestoral                23
blood sugar                 8
electrocardiographic        1
heart rate                  1
exercise induced            1
depression                  0
slope                     190
ca                        294
thal                      268
c                           0
dtype: int64
```

در این مرحله داده های مفقود را مدیریت می کنیم. برای ستون های عددی از میانه و برای ستون های غیرعددی از مد استفاده می کنیم. مقادیر مفقود در ستون های عددی با میانه و مقادیر مفقود در ستون های غیرعددی با مد پر می شوند. پس از پر کردن، تعداد داده های مفقود مجدداً بررسی می شود تا مطمئن شویم تمامی داده های مفقود برطرف شده اند.

```
# پر کردن داده های مفقود با مقدار مد و میانه در هر ستون

def handle_missing_values(df):

  # پر کردن داده های کمی با میانه
  for col in df.select_dtypes(include=['number']).columns:
    df[col] = df[col].fillna(df[col].median())

  # پر کردن داده های کیفی با مد
  for col in df.select_dtypes(include=['object']).columns:
    df[col] = df[col].fillna(df[col].mode()[0])

    return df

# اعمال تابع بر روی دیتافریم
df = handle_missing_values(df)

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

این قطعه کد شامل یک تابع برای مدیریت مقادیر گمشده در یک DataFrame و سپس اعمال این تابع بر روی دیتافریم df است. هر بخش را به تفصیل بررسی می کنیم:

- ✓ تعریف تابع handle_missing_values

این خط تابعی به نام handle_missing_values را تعریف می‌کند که یک آرگومان به نام df (که نمایانگر DataFrame است) می‌پذیرد. هدف این تابع مدیریت مقادیر گمشده در DataFrame است.

✓ پر کردن داده‌های کمی با میانه

در این بخش، ما با استفاده از select_dtypes، تمام ستون‌های عددی (کمی) DataFrame را انتخاب می‌کنیم.

کاربرد: برای هر یک از این ستون‌ها، مقادیر گمشده (NaN) با میانه آن ستون پر می‌شوند. میانه یک نماینده مرکزی از داده‌ها است و استفاده از آن به جلوگیری از تأثیر مقادیر افراطی (آوتلایرها) کمک می‌کند.

✓ پر کردن داده‌های کیفی با مد

در این بخش، تمام ستون‌های کیفی (غیر عددی) DataFrame را انتخاب می‌کنیم.

برای هر یک از این ستون‌ها، مقادیر گمشده با مد (مقدار پر تکرار) آن ستون پر می‌شوند. این روش به ما کمک می‌کند تا داده‌های گمشده را با رایج‌ترین مقدار موجود در آن ستون جایگزین کنیم.

✓ بازگشت DataFrame اصلاح‌شده

در این خط، DataFrame که مقادیر گمشده آن مدیریت شده است، به عنوان خروجی تابع بازگشت داده می‌شود.

✓ اعمال تابع بر روی دیتافریم

در این خط، تابع handle_missing_values بر روی DataFrame df اعمال می‌شود و نتیجه (DataFrame اصلاح‌شده) دوباره در متغیر df ذخیره می‌شود. این به ما اجازه می‌دهد تا تغییرات را در DataFrame اصلی اعمال کنیم.

✓ بررسی مجدد مقادیر گمشده

این خط کد تعداد مقادیر گمشده در هر ستون DataFrame را محاسبه و نمایش می‌دهد.

با استفاده از ()isnull , ما یک DataFrame جدید به دست می‌آوریم که شامل مقادیر Boolean است (True برای مقادیر گمشده و False برای مقادیر موجود). سپس با استفاده از sum()، تعداد True ها (مقادیر گمشده) برای هر ستون محاسبه می‌شود.

```
Age (age in year)        0
sex                      0
chest pain               0
blood pressure           0
cholestoral              0
blood sugar              0
electrocardiographic     0
heart rate               0
exercise induced         0
depression               0
slope                    0
ca                       0
thal                     0
c                        0
dtype: int64
```

شناسایی و مدیریت داده های تکراری

داده های تکراری به مقادیر یکسان در یک یا چند ستون از یک DataFrame اشاره دارند. شناسایی این داده ها به ما کمک می کند تا از تحلیل های نادرست و انحرافات در نتایج جلوگیری کنیم.

برای شناسایی داده های تکراری می توان از متد duplicated در کتابخانه pandas استفاده کرد. این متد به ما می گوید که کدام سطرها تکراری هستند و با مقدار True یا False مشخص می کند.

حذف داده های تکراری: می توان با استفاده از تابع drop_duplicates ، تمامی سطرهای تکراری را حذف کرد.

در این کد، سطرهای تکراری شناسایی می شوند و تعداد آن ها با استفاده از ویژگی shape نمایش داده می شود. در نهایت df جدید چاپ می شود.

```
# شناسایی داده های تکراری
df_duplicates = df.drop_duplicates()
print(df.shape, df_duplicates.shape)

duplicates = df.duplicated()
print("Duplicated Rows:")
df[duplicates]

# حذف سطرهای تکراری
df_cleaned = df.drop_duplicates()
print("\nDataFrame after removing duplicates:")
df_cleaned
```

این قطعه کد به شناسایی و حذف داده‌های تکراری در یک DataFrame پرداخته و نتایج را نمایش می‌دهد. هر بخش را به تفصیل بررسی می کنیم:

✓ شناسایی داده‌های تکراری

در این خط، ما از تابع drop_duplicates() استفاده می‌کنیم تا یک نسخه جدید از DataFrame df ایجاد کنیم که شامل سطرهای تکراری نیست. نتیجه این تابع در متغیر df_duplicates ذخیره می‌شود.

با استفاده از drop_duplicates() , ما می‌توانیم به راحتی سطرهای تکراری را حذف کنیم.

نتیجه: با استفاده از print(df.shape, df_duplicates.shape), ما ابعاد (تعداد سطرها و ستون‌ها) DataFrame اصلی و DataFrame بدون سطرهای تکراری را نمایش می‌دهیم. این کار به ما کمک می‌کند تا ببینیم چند سطر تکراری وجود داشته است.

✓ شناسایی سطرهای تکراری

در اینجا، ما از تابع duplicated() استفاده می‌کنیم که یک سری (Series) از مقادیر بولی (True/False) را برمی‌گرداند. هر مقدار True نشان‌دهنده این است که سطر مربوطه تکراری است. این تابع به ما کمک می‌کند تا سطرهای تکراری را شناسایی کنیم. با استفاده از print("Duplicated Rows:")، ما یک پیام را چاپ می‌کنیم و سپس با استفاده از df[duplicates]، سطرهای تکراری را نمایش می‌دهیم.

✓ حذف سطرهای تکراری

در این مرحله، ما دوباره از تابع drop_duplicates() استفاده می‌کنیم تا سطرهای تکراری را حذف کنیم و نتیجه را در متغیر df_cleaned ذخیره می‌کنیم. این کار به ما اجازه می‌دهد تا یک DataFrame تمیز و بدون سطرهای تکراری داشته باشیم. با استفاده از print("\nDataFrame after removing duplicates:")، ما یک پیام را چاپ می‌کنیم و سپس df_cleaned را نمایش می‌دهیم تا ببینیم که DataFrame پس از حذف سطرهای تکراری چگونه به نظر می‌رسد.

```
(597, 14) (596, 14)
Duplicated Rows:

DataFrame after removing duplicates:
```

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
0	63	1	1	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	37	1	3	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
2	41	0	2	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
3	56	1	2	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0
4	57	0	4	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0
5	57	1	4	140.0	192.0	0.0	0.0	148.0	0.0	0.4	2.0	0.0	6.0	0
6	56	0	2	140.0	294.0	0.0	2.0	153.0	0.0	1.3	2.0	0.0	3.0	0

شناسایی داده های نویز Noisy data

برای شناسایی داده های نویز، معمولاً از تحلیل آماری یا بررسی متغیرهای غیرمعمول استفاده می شود. ابتدا بررسی می کنیم که آیا در داده ها مقادیر غیرمنطقی یا نویز وجود دارد یا خیر.

این کد مقادیر منحصر به فرد هر ستون را نمایش می دهد. با این کار می توانیم مقادیر غیرمعمول یا نویز را شناسایی کنیم، مثلاً اگر ستون عددی دارای مقادیر منفی یا بیش از حد بزرگ باشد، ممکن است داده های نویزدار داشته باشیم. اگر نیاز به اصلاح باشد، در ادامه اقدام خواهیم کرد.

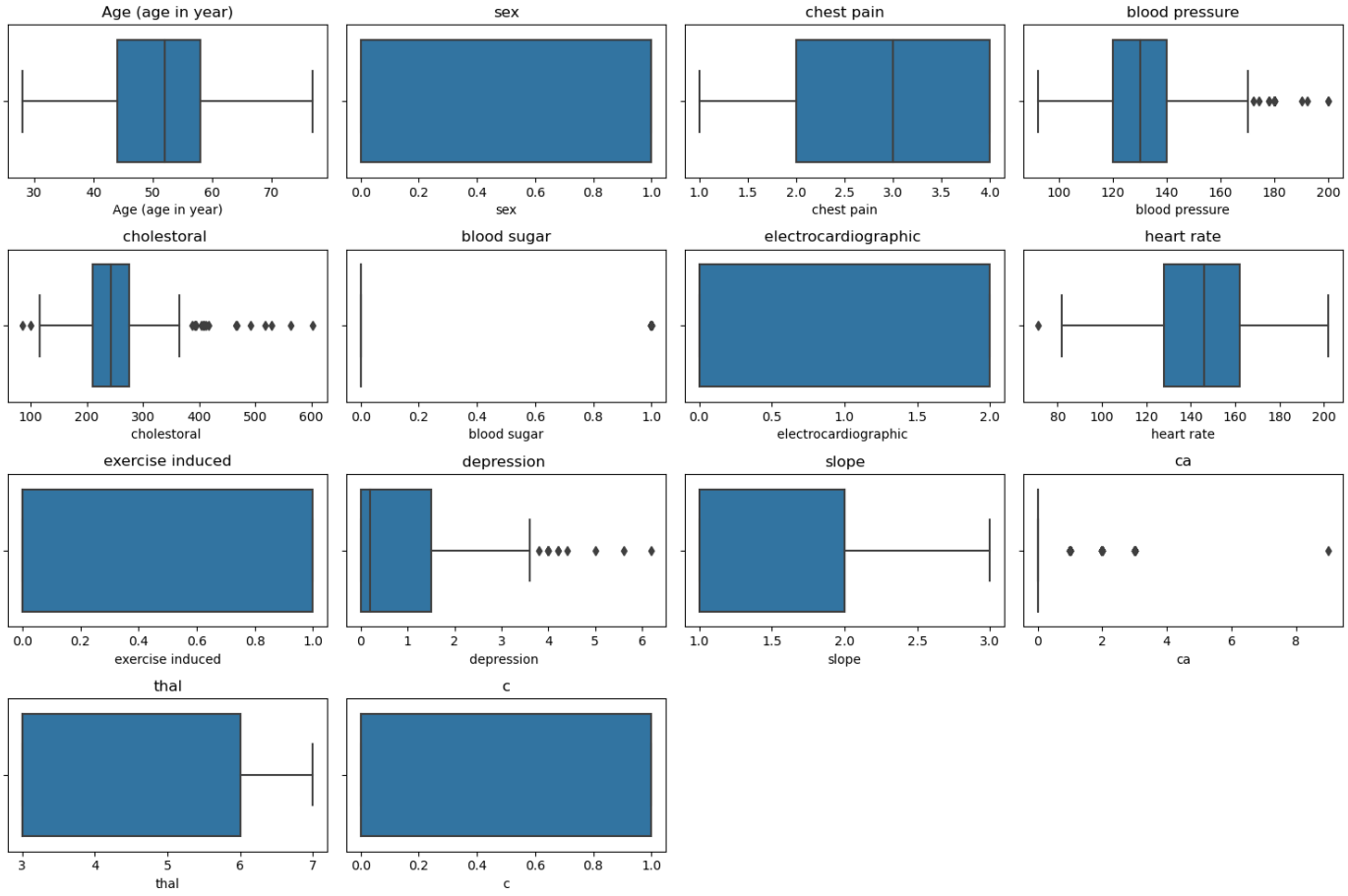
```
1 # بررسی مقادیر منحصر به فرد هر ستون برای شناسایی داده های نویز
2 for column in df.columns:
3     print(f"{column}: {df[column].unique()}")
```

نمایش **boxplot** برای تحلیل وضعیت داده ها و شروع مدیریت داده های پرت

با استفاده از نمودارهای زیر ستون هایی که داده های پرت تاثیرگذار دارن را مشخص میکنیم و در مرحله بعد آن ها را مدیریت می کنیم.

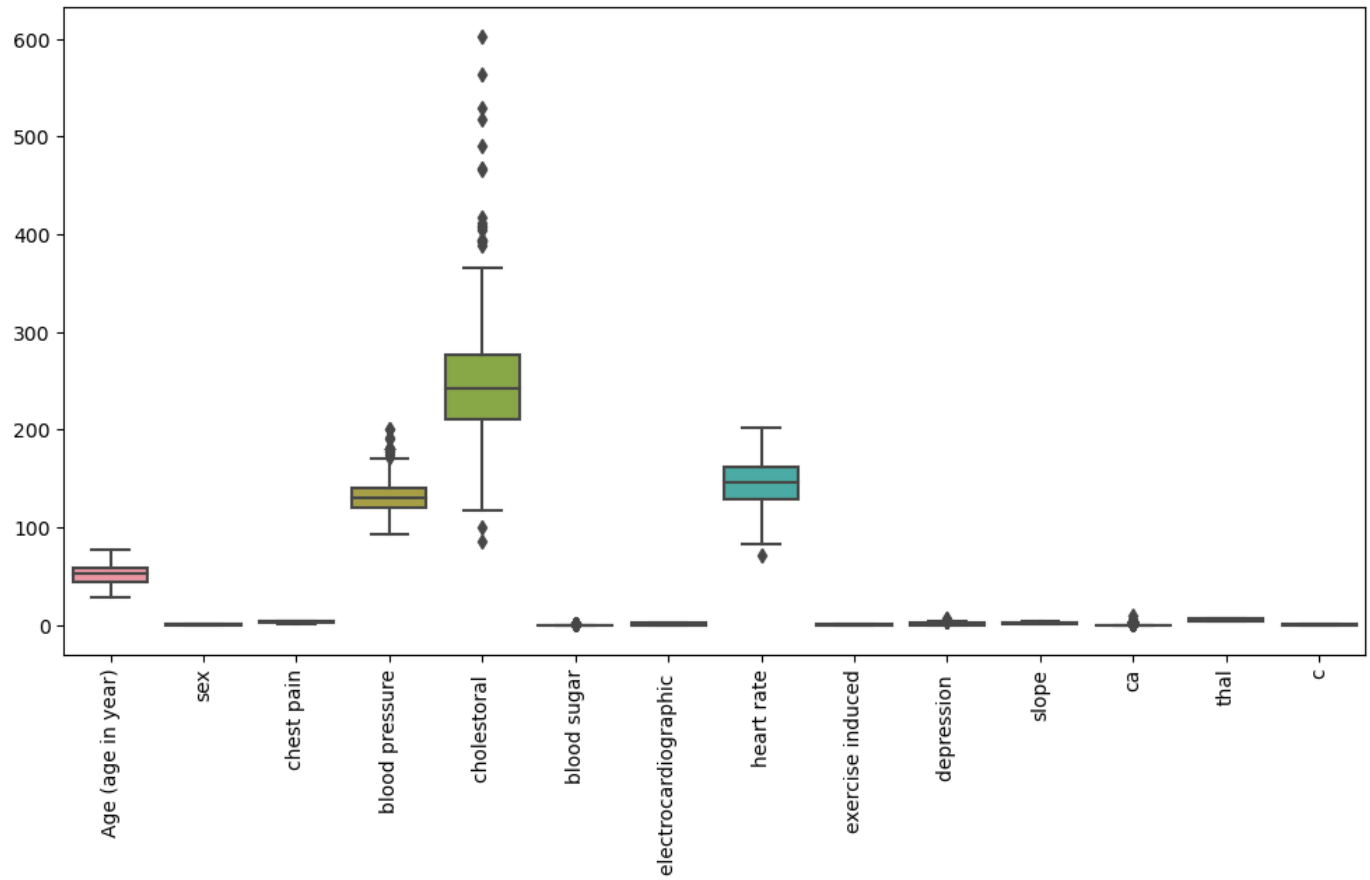
روش اول

```python
# 1 ایجاد باکس پلات برای همه ستون های عددی
plt.figure(figsize=(15, 10))
for i, col in enumerate(df.select_dtypes(include=['number']).columns):
  plt.subplot(4, 4, i+1)
  sns.boxplot(x=df[col])
  plt.title(col)
plt.tight_layout()
plt.show()
```



روش دوم

```python
# 2 ایجاد باکس پلات برای همه ستون های عددی
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.show()
```



شناسایی و مدیریت داده های پرت Outliers data

در این مرحله، ما به شناسایی و مدیریت داده‌های پرت می‌پردازیم. داده های پرت را با استفاده از روش IQR (Interquartile Range) شناسایی می کنیم. داده های پرت می توانند نتایج مدل سازی را تحت تأثیر قرار دهند، به همین دلیل آن ها را به صورت جداگانه بررسی می کنیم. بعضی از ستون های خاص را با میانه و داده های پرت سایر ستون ها را با میانگین پر می کنیم.

✓ محاسبه Q1، Q3 و IQR

```
# محاسبه Q1، Q3 و IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

در این بخش، ما مقادیر Q1 (نمره چهارم) و Q3 (نمره سوم) را برای هر ستون در DataFrame محاسبه می‌کنیم. Q1 نشان‌دهنده ۲۵ درصد پایین داده‌ها و Q3 نشان‌دهنده ۷۵ درصد بالا است. IQR (Interquartile Range): تفاوت بین Q3 و Q1 است که نشان‌دهنده دامنه میانه داده‌ها می‌باشد.

حد پایین و بالا: با استفاده از ۱.۵ برابر IQR، ما حدود پایین و بالای داده‌های معمولی را تعیین می‌کنیم. هر مقداری که از این حدود خارج باشد به عنوان داده پرت شناسایی می‌شود.

✓ شناسایی داده‌های پرت

```
# شناسایی داده های پرت
outliers = df[(df < lower_bound) | (df > upper_bound)].dropna(how='all')
print("Number of All outliers detected:", outliers.shape[0])
```

در اینجا، ما داده‌های پرت را شناسایی می‌کنیم. با استفاده از شرط (df < lower_bound) | (df > upper_bound)، هر مقداری که خارج از حدود تعیین‌شده باشد شناسایی می‌شود.

dropna(how='all') برای حذف سطرهایی است که تمام مقادیر آن‌ها ناپیدا (NaN) هستند.

در نتیجه تعداد کل داده‌های پرت شناسایی‌شده چاپ می‌شود.

✓ شناسایی داده‌های پرت در ستون‌های خاص

```
# شناسایی داده های پرت در ستون های خاص
outliers_specific = df[
    (df['blood pressure'] < (Q1['blood pressure'] - 1.5 * IQR['blood pressure'])) |
    (df['blood pressure'] > (Q3['blood pressure'] + 1.5 * IQR['blood pressure'])) |
    (df['cholestoral '] < (Q1['cholestoral '] - 1.5 * IQR['cholestoral '])) |
    (df['cholestoral '] > (Q3['cholestoral '] + 1.5 * IQR['cholestoral '])) |
    (df['depression '] < (Q1['depression '] - 1.5 * IQR['depression '])) |
    (df['depression '] > (Q3['depression '] + 1.5 * IQR['depression '])) |
    (df['heart rate'] < (Q1['heart rate'] - 1.5 * IQR['heart rate'])) |
    (df['heart rate'] > (Q3['heart rate'] + 1.5 * IQR['heart rate']))
]
print("Number of outliers detected in specific columns :", outliers_specific.shape[0])
```

در این بخش، ما به شناسایی داده‌های پرت در چهار ستون خاص (فشار خون، کلسترول، افسردگی و ضربان قلب) می‌پردازیم.

✓ کپی از داده‌های اصلی برای مدیریت داده‌های پرت

```
# کپی از داده های اصلی برای مدیریت داده های پرت
data_no_outliers = df.copy()
```

در این خط، یک کپی از DataFrame اصلی ایجاد می‌کنیم تا بتوانیم تغییرات را بدون تأثیر بر داده‌های اصلی انجام دهیم.

✓ پر کردن مقادیر گمشده در ستون‌های خاص

```
# ستون های خاص با میانه پر می شوند
special_columns = ['blood pressure', 'cholestoral ', 'depression ', 'heart rate']
data_no_outliers[special_columns] = data_no_outliers[special_columns].fillna(data_no_outliers[special_columns].median())
```

در این بخش، ما مقادیر گمشده در ستون‌های خاص را با میانه (median) آن ستون‌ها پر می‌کنیم. این روش معمولاً برای داده‌های عددی مناسب است.

✓ پر کردن سایر مقادیر گمشده با میانگین

```
# سایر ستون ها با میانگین پر می شوند
data_no_outliers = data_no_outliers.fillna(data_no_outliers.mode())
```

در این خط، ما مقادیر گمشده در سایر ستون‌ها را با مد (mode) آن ستون‌ها پر می‌کنیم. مد به معنای رایج ترین مقدار در یک ستون است.

✓ نمایش داده‌های بدون پرت و پر شده

```
# نمایش داده های بدون پرت و پر شده
print("Data without outliers and filled missing values:")
data_no_outliers
```

در این بخش، ما داده‌های نهایی (بدون داده‌های پرت و با مقادیر گمشده پر شده) را نمایش می‌دهیم.

```
Number of All outliers detected: 196
Number of outliers detected in specific columns : 44
Data without outliers and filled missing values:
```

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
0	63	1	1	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	37	1	3	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
2	41	0	2	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
3	56	1	2	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0
4	57	0	4	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0
5	57	1	4	140.0	192.0	0.0	0.0	148.0	0.0	0.4	2.0	0.0	6.0	0

انتخاب ویژگی ها و برچسب ها

در این مرحله، داده ها به دو بخش تقسیم می شوند: ویژگی ها (X) و متغیر هدف (y) . این کار برای آماده سازی داده ها برای آموزش مدل ها ضروری است. متغیر هدف نشان می دهد که آیا فرد دچار سکته قلبی شده است یا خیر.

ستون target به عنوان متغیر وابسته در نظر گرفته می شود و سایر ستون ها به عنوان متغیرهای مستقل (ویژگی ها) برای مدل سازی استفاده خواهند شد.

```
1 # تقسیم داده ها به ویژگی ها و هدف
2 X = df.drop('c', axis=1)
3 y = df['c']
```

تقسیم داده ها به مجموعه های آموزش ، آزمون و اعتبارسنجی

در این مرحله داده ها به مجموعه های آموزش ، آزمون و اعتبارسنجی تقسیم می شوند.

- مجموعه آموزشی 70% :(Training Set) داده ها برای آموزش مدل استفاده می شود.
- مجموعه اعتبارسنجی 15% :(Validation Set) از داده ها برای تنظیم هایپرپارامترها و ارزیابی مدل استفاده می شود.
- مجموعه آزمون 15% :(Test Set) از داده ها برای ارزیابی نهایی عملکرد مدل استفاده می شود.

```python
from sklearn.model_selection import train_test_split

# تقسیم داده ها به مجموعه های آموزشی، آزمون و اعتبارسنجی

# ابتدا تقسیم داده ها به مجموعه آموزشی و آزمون
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# سپس تقسیم مجموعه آموزشی و اعتبارسنجی
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=42)

# چاپ اندازه مجموعه ها
print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

این قطعه کد مربوط به تقسیم داده‌ها به مجموعه‌های آموزشی، آزمون و اعتبارسنجی است که یکی از مراحل کلیدی در فرآیند آموزش مدل‌های یادگیری ماشین به شمار می‌آید. هر بخش را به تفصیل بررسی کنیم:

✓ وارد کردن تابع train_test_split

در این خط، ما تابع train_test_split را از کتابخانه sklearn.model_selection وارد می‌کنیم. این تابع برای تقسیم داده‌ها به مجموعه‌های مختلف استفاده می‌شود. train_test_split به ما این امکان را می‌دهد که داده‌ها را به راحتی به مجموعه‌های آموزشی و آزمون تقسیم کنیم، که این کار برای ارزیابی عملکرد مدل‌های یادگیری ماشین ضروری است.

✓ تقسیم داده‌ها به مجموعه‌های آموزشی و آزمون

در این خط، ما داده‌ها را به دو بخش تقسیم می‌کنیم: مجموعه آموزشی و اعتبارسنجی (X_train_val و y_train_val) و مجموعه آزمون (X_test و y_test).

پارامترها:

- X: ویژگی‌های داده‌ها (مجموعه‌ای از متغیرهای مستقل).
- y: برچسب‌ها یا خروجی‌ها (مجموعه‌ای از متغیرهای وابسته).
- test_size=0.2: 20 درصد از داده‌ها برای مجموعه آزمون اختصاص داده می‌شود.
- random_state=42: این پارامتر برای تنظیم تصادفی بودن تقسیم‌بندی استفاده می‌شود، که باعث می‌شود نتایج قابل تکرار باشند.

در نتیجه پس از این خط، ما دو مجموعه داریم: X_train_val و y_train_val برای آموزش و اعتبارسنجی و X_test و y_test برای آزمون.

✓ تقسیم مجموعه آموزشی و اعتبارسنجی

در این مرحله، ما مجموعه آموزشی و اعتبارسنجی را از X_train_val و y_train_val تقسیم می‌کنیم.

پارامترها:

- test_size=0.25: 25 درصد از داده‌های X_train_val و y_train_val برای مجموعه اعتبارسنجی اختصاص داده می‌شود. این بدان معناست که ۷۵ درصد از داده‌ها به مجموعه آموزشی (X_train و y_train) و ۲۵ درصد به مجموعه اعتبارسنجی (X_val و y_val) اختصاص می‌یابد.

اکنون ما سه مجموعه داریم: X_train و y_train برای آموزش، X_val و y_val برای اعتبارسنجی، و X_test و y_test برای آزمون.

✓ چاپ اندازه مجموعه‌ها

در این بخش، اندازه هر یک از مجموعه‌ها را چاپ می‌کنیم. با استفاده از shape[۰]، تعداد سطرها (نمونه‌ها) در هر مجموعه را به دست می‌آوریم و آن‌ها را به صورت خوانا چاپ می‌کنیم. این کار به ما کمک می‌کند تا مطمئن شویم که داده‌ها به درستی تقسیم شده‌اند.

```
Training set size: 357
Validation set size: 120
Test set size: 120
```

مدیریت مجدد مقادیر گمشده پس از اعمال تغییرات

گاهی بعد از اجرای دستورات در راستای پاکسازی داده ها شاهد خالی شدن (ایجاد داده های مفقوده) دیتا در بعضی از سطرها می شویم. به همین دلیل قبل از نرمال سازی و استانداردسازی مجدد داده های مفقوده را بررسی می کنیم.

```
1 from sklearn.impute import SimpleImputer
2
3 # مدیریت مقادیر گمشده
4 imputer = SimpleImputer(strategy='mean')
5 X_train_imputed = imputer.fit_transform(X_train)
6 X_test_imputed = imputer.transform(X_test)
```

این قطعه کد به مدیریت مقادیر گمشده در داده‌ها می‌پردازد و از کتابخانه scikit-learn برای این کار استفاده می‌کند. در ادامه، هر بخش از این کد را به تفصیل بررسی می‌کنیم:

✓ وارد کردن کتابخانه

در این خط، ما کلاس SimpleImputer را از ماژول sklearn.impute وارد می‌کنیم. SimpleImputer ابزاری است که به ما اجازه می‌دهد تا مقادیر گمشده در داده‌ها را با استفاده از استراتژی‌های مختلف پر کنیم. یکی از این استراتژی‌ها، پر کردن مقادیر گمشده با میانگین (mean) است.

✓ مدیریت مقادیر گمشده

در این خط، یک شیء از کلاس SimpleImputer ایجاد می‌کنیم و استراتژی پر کردن را به عنوان mean تعیین می‌کنیم. با انتخاب استراتژی mean, هر مقدار گمشده در ویژگی‌ها با میانگین آن ویژگی پر خواهد شد. این روش به ویژه زمانی مفید است که داده‌ها به طور تقریبی نرمال توزیع شده باشند.

✓ پر کردن مقادیر گمشده در مجموعه آموزش

در این خط، ما از متد fit_transform() استفاده می‌کنیم تا مقادیر گمشده در مجموعه آموزش (X_train) را شناسایی و با میانگین پر کنیم.

- fit() : این متد میانگین هر ویژگی را محاسبه می‌کند.
- transform() : این متد مقادیر گمشده را با میانگین‌های محاسبه شده پر می‌کند.

در نتیجه X_train_imputed شامل داده‌های آموزش است که مقادیر گمشده آن پر شده‌اند.

✓ پر کردن مقادیر گمشده در مجموعه آزمون

در این خط، ما از متد transform() استفاده می‌کنیم تا مقادیر گمشده در مجموعه آزمون (X_test) را پر کنیم.

در اینجا، از همان میانگین‌هایی که در مرحله قبل برای مجموعه آموزش محاسبه شده‌اند، استفاده می‌شود. این کار مهم است تا اطمینان حاصل کنیم که داده‌های آزمون با همان پارامترهایی که بر اساس داده‌های آموزش محاسبه شده‌اند، پردازش می‌شوند.

در نتیجه X_test_imputed شامل داده‌های آزمون است که مقادیر گمشده آن نیز پر شده‌اند.

نرمال سازی و استانداردسازی

برای بهبود عملکرد مدل های مختلف، داده ها را نرمال سازی و استانداردسازی می کنیم تا تمامی ویژگی ها به یک مقیاس مشترک برسند. در این مرحله، داده ها با استفاده از StandardScaler نرمال سازی می شوند. نرمال سازی به مدل کمک می کند تا بهتر یاد بگیرد، به ویژه در مدل هایی که به مقیاس داده ها حساس هستند.

```python
from sklearn.preprocessing import StandardScaler

# استانداردسازی داده ها
scaler = StandardScaler()

# استانداردسازی مجموعه آموزشی
X_train_scaled = scaler.fit_transform(X_train)

# استانداردسازی مجموعه اعتبارسنجی
X_val_scaled = scaler.transform(X_val)

# استانداردسازی مجموعه آزمون
X_test_scaled = scaler.transform(X_test)

# نمایش اطلاعات
print(f"Scaled Training set shape: {X_train_scaled.shape}")
print(f"Scaled Validation set shape: {X_val_scaled.shape}")
print(f"Scaled Test set shape: {X_test_scaled.shape}")
```

این قطعه کد به استانداردسازی داده‌ها در یک مدل یادگیری ماشین می‌پردازد. استانداردسازی یک تکنیک مهم است که به مقیاس‌دهی ویژگی‌ها به یک توزیع نرمال (معمولاً با میانگین صفر و انحراف معیار یک) کمک می‌کند. هر بخش از این کد را به تفصیل بررسی کنیم:

✓ وارد کردن کتابخانه

در این خط، ما StandardScaler را از کتابخانه sklearn.preprocessing وارد می‌کنیم. این کلاس برای استانداردسازی داده‌ها استفاده می‌شود.

✓ ایجاد شیء StandardScaler

در این خط، یک شیء از کلاس StandardScaler ایجاد می‌کنیم. این شیء برای محاسبه میانگین و انحراف معیار از داده‌های آموزشی استفاده می‌شود و سپس این مقادیر برای مقیاس‌دهی داده‌های آموزشی و سایر مجموعه‌ها به کار می‌روند.

✓ استانداردسازی مجموعه آموزشی

در این خط، از متد fit_transform برای استانداردسازی مجموعه آموزشی X_train استفاده می‌کنیم.

- fit: مقادیر میانگین و انحراف معیار را از داده‌های آموزشی محاسبه می‌کند.
- transform: داده‌ها را بر اساس این مقادیر مقیاس‌دهی می‌کند.

در نتیجه داده‌های استانداردشده در متغیر X_train_scaled ذخیره می‌شوند.

✓ استانداردسازی مجموعه اعتبارسنجی

در این خط، از متد transform برای استانداردسازی مجموعه اعتبارسنجی X_val استفاده می‌کنیم.

در اینجا، ما فقط از transform استفاده می‌کنیم زیرا نمی‌خواهیم مقادیر میانگین و انحراف معیار را دوباره محاسبه کنیم. به جای آن، از مقادیر محاسبه‌شده از مجموعه آموزشی استفاده می‌کنیم.

✓ استانداردسازی مجموعه آزمون

مشابه مرحله قبلی، در این خط، از متد transform برای استانداردسازی مجموعه آزمون X_test استفاده می‌کنیم.

مانند مجموعه اعتبارسنجی، ما از مقادیر محاسبه‌شده از مجموعه آموزشی برای استانداردسازی داده‌های آزمون استفاده می‌کنیم.

✓ نمایش اطلاعات

در این بخش، با استفاده از تابع print, ابعاد (شکل) مجموعه‌های استانداردشده را نمایش می‌دهیم. این اطلاعات به ما کمک می‌کند تا تأیید کنیم که داده‌ها به درستی استانداردسازی شده‌اند و ابعاد آن‌ها مطابق با انتظار است.

```
Scaled Training set shape: (357, 13)
Scaled Validation set shape: (120, 13)
Scaled Test set shape: (120, 13)
```

مدل سازی و ارزیابی مدل ها

در این مرحله، ما چندین مدل مختلف را تعریف و آموزش می دهیم. مدل های انتخاب شده شامل:

✓ Logistic Regression: مدل رگرسیون لجستیک برای پیش‌بینی احتمال وقوع سکته قلبی.
✓ K-Nearest Neighbors (KNN): یک الگوریتم مبتنی بر همسایگی که برای پیش‌بینی کلاس‌ها استفاده می‌شود.
✓ Decision Tree: یک مدل درخت تصمیم که تصمیمات را بر اساس ویژگی‌ها می‌گیرد.
✓ Random Forest: یک مدل جنگل تصادفی که ترکیبی از چندین درخت تصمیم است.
✓ Support Vector Machine (SVM): یک الگوریتم که به یافتن بهترین مرز جداسازی بین کلاس‌ها می‌پردازد.
✓ Neural Network: یک شبکه عصبی چندلایه که می‌تواند الگوهای پیچیده را یاد بگیرد.

پس از آموزش هر مدل، دقت آن‌ها با استفاده از مجموعه اعتبارسنجی محاسبه می‌شود. در نهایت دقت مدل ها را با هم مقایسه و بهترین مدل را انتخاب می کنیم.

وارد کردن کتابخانه ها

```python
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, precision_recall_curve
```

مدل سازی

```
# تعریف مدل ها
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machine": SVC(probability=True),
    "Neural Network": MLPClassifier(max_iter=1000)
}

# آموزش و ارزیابی مدل ها
results = {}

for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)  # آموزش مدل
    y_test_pred = model.predict(X_test_scaled)  # پیش بینی بر روی مجموعه اعتبارسنجی
    accuracy = accuracy_score(y_test, y_test_pred)  # محاسبه دقت
    results[model_name] = accuracy
#     print(f"Model: {model_name}, Validation Accuracy: {accuracy:.4f}")
```

این قطعه کد به تعریف و ارزیابی چندین مدل یادگیری ماشین برای پیش‌بینی بر اساس داده‌های آموزشی و اعتبارسنجی می‌پردازد. هر بخش را به تفصیل بررسی کنیم:

✓ تعریف مدل‌ها

در این بخش، یک دیکشنری به نام models تعریف می‌شود که شامل چندین مدل یادگیری ماشین است. هر مدل با نام خود به عنوان کلید (key) و شیء مدل به عنوان مقدار (value) ذخیره می‌شود.

نکته: برای مدل Random Forest، random_state=42 تعیین شده است تا نتایج قابل تکرار باشند. همچنین، برای SVC، probability=True تنظیم شده است تا احتمال پیش‌بینی‌ها محاسبه شود.

✓ آموزش و ارزیابی مدل‌ها

در این بخش، یک دیکشنری به نام results برای ذخیره دقت هر مدل ایجاد می‌شود.

- حلقه: با استفاده از یک حلقه for، به ازای هر مدل در دیکشنری models:
- آموزش مدل: model.fit(X_train_scaled, y_train) مدل با داده‌های آموزشی (X_train_scaled و y_train) آموزش داده می‌شود. X_train_scaled معمولاً نشان‌دهنده ویژگی‌های مقیاس شده است.
- پیش‌بینی: y_test_pred = model.predict(X_test_scaled) پیش‌بینی‌هایی برای مجموعه اعتبارسنجی (X_test_scaled) انجام می‌شود.
- محاسبه دقت: accuracy = accuracy_score(y_test, y_test_pred) دقت پیش‌بینی‌ها با استفاده از تابع accuracy_score محاسبه می‌شود. y_test شامل برچسب‌های واقعی برای مجموعه اعتبارسنجی است.
- ذخیره نتایج: دقت هر مدل در دیکشنری results ذخیره می‌شود.

✓ (اختیاری) چاپ نتایج

این خط کد (که در حال حاضر کامنت شده است) به چاپ نام مدل و دقت آن کمک می‌کند. اگر این خط را از حالت کامنت خارج کنید، نتایج دقت هر مدل به صورت فرمت شده با چهار رقم اعشار نمایش داده خواهد شد.

بررسی اورفیت و دقت مدل ها

در این مرحله، از K-Fold Cross-Validation برای ارزیابی مدل ها استفاده می شود. داده ها به K بخش تقسیم می شوند و هر مدل K بار آموزش داده می شود. این کار به ما کمک می کند تا دقت مدل ها را به طور دقیق تری ارزیابی کنیم و از اورفیت جلوگیری کنیم.

```
#  استفاده از K-Fold Cross-Validation برای ارزیابی مدل ها
kf = KFold(n_splits=5, shuffle=True, random_state=42)

cv_results = {}

for model_name, model in models.items():
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=kf, scoring='accuracy')
    cv_results[model_name] = cv_scores.mean()
#     print(f"Model: {model_name}, Cross-Validation Score: {cv_results[model_name]:.4f}")
```

این قطعه کد به ارزیابی مدل‌های مختلف یادگیری ماشین با استفاده از روش K-Fold Cross-Validation می‌پردازد. در ادامه، هر بخش از این کد را به تفصیل توضیح می‌دهیم:

✓ تعریف K-Fold Cross-Validation

در این خط، ما یک شیء از کلاس KFold از کتابخانه sklearn.model_selection ایجاد می‌کنیم.

پارامترها:

- n_splits=5: تعداد تقسیمات (folds) را مشخص می‌کند. در اینجا، داده‌ها به ۵ بخش تقسیم می‌شوند.
- shuffle=True: این گزینه مشخص می‌کند که داده‌ها قبل از تقسیم‌بندی به صورت تصادفی جابجا شوند. این کار به جلوگیری از بایاس در انتخاب داده‌ها کمک می‌کند.
- random_state=42: این پارامتر برای تضمین قابلیت تکرار استفاده می‌شود. با تعیین یک عدد خاص، هر بار که کد اجرا شود، همان تقسیمات تصادفی ایجاد می‌شود.

✓ ایجاد دیکشنری برای ذخیره نتایج

در این خط، یک دیکشنری خالی به نام cv_results ایجاد می‌کنیم. این دیکشنری برای ذخیره نتایج ارزیابی مدل‌ها استفاده خواهد شد.

✓ ارزیابی مدل‌ها

در این بخش، ما از یک حلقه for برای ارزیابی هر مدل استفاده می‌کنیم. فرض بر این است که models یک دیکشنری است که نام مدل‌ها به عنوان کلید و شیء مدل‌های یادگیری ماشین به عنوان مقادیر را ذخیره می‌کند.

عملیات:

- cross_val_score(...) : این تابع از کتابخانه sklearn.model_selection برای انجام K-Fold Cross-Validation بر روی هر مدل استفاده می‌شود.
- Model : مدل فعلی که در حال ارزیابی است.
- X_train_scaled : ویژگی‌های آموزش که پیش از این مقیاس‌بندی شده‌اند.
- y_train : برچسب‌های آموزش.
- cv=kf : مشخص می‌کند که از تقسیمات K-Fold تعریف‌شده استفاده شود.
- scoring=' accuracy ' : معیاری که برای ارزیابی مدل استفاده می‌شود. در اینجا، دقت (accuracy) انتخاب شده است.
- cv_scores.mean(): میانگین نمرات ارزیابی برای مدل فعلی محاسبه می‌شود و در دیکشنری cv_results با نام مدل ذخیره می‌شود.

✓ (اختیاری) چاپ نتایج

این خط (که به صورت کامنت درآمده است) می‌تواند برای چاپ نتایج ارزیابی هر مدل استفاده شود. با حذف علامت # این خط می‌تواند به شما کمک کند تا نتایج را در کنسول مشاهده کنید.

مقایسه دقت مدل ها

با جمع آوری نتایج مدل ها و تبدیل آن ها به یک جدول و مرتب سازی بر اساس بیشترین مقدار میتوانیم مدل های بهینه تر را نمایش و انتخاب کنیم. همانطور که در جدول زیر مشخص شده است مدل های Support Vector Machine و Logistic Regressionو Random Forest بهترین دقت را داشته اند.

```
# نمایش نتایج اعتبارسنجی
results_df = pd.DataFrame(results.items(), columns=['Model', 'Validation Accuracy'])
results_df.sort_values(by='Validation Accuracy', ascending=False, inplace=True)

# نمایش نتایج اعتبارسنجی متقاطع
cv_results_df = pd.DataFrame(cv_results.items(), columns=['Model', 'Cross-Validation Score'])
cv_results_df.sort_values(by='Cross-Validation Score', ascending=False, inplace=True)

# نهایی DataFrame در یک DataFrame ترکیب دو
final_results_df = pd.merge(results_df, cv_results_df, on='Model', how='outer')

# نمایش جدول نهایی
final_results_df
```

این قطعه کد به نمایش نتایج اعتبارسنجی مدل‌های یادگیری ماشین می‌پردازد و نتایج را به صورت یک جدول نهایی ترکیب می‌کند. هر بخش را به تفصیل بررسی می کنیم:

✓ ایجاد DataFrame برای نتایج اعتبارسنجی

در این خط، ما یک DataFrame به نام results_df ایجاد می‌کنیم که شامل نتایج اعتبارسنجی مدل‌ها است.

- results.items() فرض می‌کند که results یک دیکشنری است که نام مدل‌ها به عنوان کلید و دقت اعتبارسنجی (Validation Accuracy) به عنوان مقدار دارد.
- با استفاده از pd.DataFrame() , این دیکشنری به یک DataFrame تبدیل می‌شود و دو ستون به نام‌های 'Model' و 'Validation Accuracy' ایجاد می‌شود.
- مرتب‌سازی: سپس با استفاده از DataFrame ,sort_values() بر اساس دقت اعتبارسنجی به صورت نزولی مرتب می‌شود (ascending=False).

✓ ایجاد DataFrame برای نتایج اعتبارسنجی متقاطع

در این خط، ما یک DataFrame به نام cv_results_df ایجاد می‌کنیم که شامل نتایج اعتبارسنجی متقاطع (Cross-Validation) مدل‌ها است.

- cv_results.items() فرض می‌کند که cv_results یک دیکشنری است که نام مدل‌ها به عنوان کلید و نمره اعتبارسنجی متقاطع به عنوان مقدار دارد.
- مانند مرحله قبل، این دیکشنری به یک DataFrame تبدیل می‌شود و دو ستون به نام‌های 'Model' و 'Cross-Validation Score' ایجاد می‌شود.
- مرتب‌سازی: سپس DataFrame بر اساس نمره اعتبارسنجی متقاطع به صورت نزولی مرتب می‌شود.

✓ ترکیب دو DataFrame در یک DataFrame نهایی

در این خط، ما دو DataFrame results_df و cv_results_df را با استفاده از تابع merge() ترکیب می‌کنیم.

- on='Model' : این گزینه مشخص می‌کند که ترکیب بر اساس ستون 'Model' انجام شود.
- how='outer' : این گزینه تعیین می‌کند که از نوع ادغام بیرونی (outer join) استفاده شود، به این معنی که تمام مدل‌ها از هر دو DataFrame در نتیجه نهایی قرار می‌گیرند، حتی اگر در یکی از DataFrameها موجود نباشند.

✓ نمایش جدول نهایی

در این مرحله، DataFrame نهایی final_results_df که شامل نتایج اعتبارسنجی و اعتبارسنجی متقاطع برای هر مدل است، نمایش داده می‌شود. این جدول به ما امکان می‌دهد تا به راحتی عملکرد مدل‌های مختلف را مقایسه کنیم.

	Model	Validation Accuracy	Cross-Validation Score
0	Support Vector Machine	0.841667	0.828912
1	Neural Network	0.841667	0.770031
2	Random Forest	0.833333	0.800978
3	Logistic Regression	0.825000	0.820618
4	K-Nearest Neighbors	0.808333	0.826213
5	Decision Tree	0.800000	0.728169

استفاده از معیارهای ارزیابی مختلف

در این مرحله، از معیارهای ارزیابی مختلف برای سنجش عملکرد مدل‌ها استفاده می‌شود:

- دقت(Accuracy) : نسبت پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها.
- F1 Score: میانگین هارمونیک دقت و یادآوری.
- AUC-ROC: ارزیابی عملکرد مدل بر اساس منحنی دریافت – عملکرد (Receiver Operating Characteristic) .

این معیارها به ما کمک می‌کنند تا عملکرد مدل‌ها را به طور جامع‌تری ارزیابی کنیم.

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# فرض کنید X_train_scaled و X_test_scaled آرایه های NumPy هستند
X_train_scaled = pd.DataFrame(X_train_scaled)
X_test_scaled = pd.DataFrame(X_test_scaled)

# بررسی مقادیر گم شده در مجموعه داده های آموزشی و آزمون
print("Missing values in X_train_scaled:")
print(pd.isnull(X_train_scaled).sum())

print("Missing values in X_test_scaled:")
print(pd.isnull(X_test_scaled).sum())

# پر کردن مقادیر گم شده با میانگین یا میانه
X_train_scaled.fillna(X_train_scaled.mean(), inplace=True)
X_test_scaled.fillna(X_test_scaled.mean(), inplace=True)
```

این قطعه کد به بررسی و مدیریت مقادیر گمشده در مجموعه‌های داده آموزشی و آزمون پرداخته و همچنین از کتابخانه sklearn.metrics برای ارزیابی مدل‌های یادگیری ماشین استفاده می‌کند. هر بخش را به تفصیل بررسی می کنیم:

✓ وارد کردن توابع ارزیابی

در این خط، توابع accuracy_score, f1_score, و roc_auc_score از کتابخانه sklearn.metrics وارد می‌شوند. این توابع برای ارزیابی عملکرد مدل‌های یادگیری ماشین استفاده می‌شوند.

- accuracy_score: دقت مدل را محاسبه می‌کند، که نسبت تعداد پیش‌بینی‌های صحیح به کل تعداد پیش‌بینی‌ها است.
- f1_score: میانگین هارمونیک دقت و یادآوری (recall) را محاسبه می‌کند و برای ارزیابی مدل‌هایی که با داده‌های نامتوازن کار می‌کنند، مفید است.
- roc_auc_score: مساحت زیر منحنی ROC (Receiver Operating Characteristic) را محاسبه می‌کند که نشان‌دهنده توانایی مدل در تفکیک بین کلاس‌ها است.

✓ تبدیل آرایه‌های NumPy به DataFrame

در اینجا، فرض می‌شود که X_train_scaled و X_test_scaled آرایه‌های NumPy هستند. ما آن‌ها را به DataFrame تبدیل می‌کنیم تا بتوانیم از قابلیت‌های Pandas برای مدیریت داده‌ها استفاده کنیم. این کار به ما اجازه می‌دهد تا به راحتی با داده‌ها کار کنیم و از توابع Pandas استفاده کنیم.

✓ بررسی مقادیر گمشده در مجموعه‌های داده آموزشی و آزمون

در این بخش، ما به بررسی مقادیر گمشده (NaN) در DataFrame های X_train_scaled و X_test_scaled می‌پردازیم.

- pd.isnull(X_train_scaled).sum() : این خط تعداد مقادیر گمشده در هر ستون DataFrame X_train_scaled را محاسبه می‌کند و نمایش می‌دهد.
- pd.isnull(X_test_scaled).sum() : به طور مشابه، این خط تعداد مقادیر گمشده در DataFrame X_test_scaled را محاسبه و نمایش می‌دهد.

در نتیجه این اطلاعات به ما کمک می‌کند تا ببینیم آیا نیاز به پر کردن مقادیر گمشده داریم یا خیر.

✓ پر کردن مقادیر گمشده با میانگین

در اینجا، ما مقادیر گمشده را با میانگین مقادیر هر ستون پر می‌کنیم.

- X_train_scaled.fillna(X_train_scaled.mean(), inplace=True): این خط مقادیر گمشده در DataFrame X_train_scaled را با میانگین ستون‌های مربوطه پر می‌کند.
- X_test_scaled.fillna(X_test_scaled.mean(), inplace=True): به طور مشابه، این خط مقادیر گمشده در DataFrame X_test_scaled را با میانگین ستون‌های مربوطه پر می‌کند.

پر کردن مقادیر گمشده با میانگین یک روش متداول است که به ما کمک می‌کند تا از تأثیر منفی مقادیر گمشده بر روی مدل جلوگیری کنیم.

```
Missing values in X_train_scaled:
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
dtype: int64
```

```
Missing values in X_test_scaled:
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
dtype: int64
```

پیش بینی بر روی مجموعه آزمون و محاسبه معیارهای ارزیابی

```python
# پیش بینی بر روی مجموعه آزمون و محاسبه معیارهای ارزیابی
test_results = {}

for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)  # آموزش مجدد مدل بر روی مجموعه آموزشی
    y_test_pred = model.predict(X_test_scaled)  # پیش بینی بر روی مجموعه آزمون
    accuracy = accuracy_score(y_test, y_test_pred)
    f1 = f1_score(y_test, y_test_pred)
    roc_auc = roc_auc_score(y_test, model.predict_proba(X_test_scaled)[:, 1])
    test_results[model_name] = {
        'Accuracy': accuracy,
        'F1 Score': f1,
        'AUC-ROC': roc_auc
    }

# نمایش نتایج
pd.DataFrame(test_results).T
```

این قطعه کد به ما امکان می‌دهد تا مدل‌های یادگیری ماشین را بر روی مجموعه آزمون اجرا کنیم و معیارهای ارزیابی مختلفی را محاسبه کنیم. هر بخش از کد را به تفصیل بررسی می کنیم:

✓ آماده‌سازی دیکشنری برای ذخیره نتایج

در این خط، یک دیکشنری خالی به نام test_results ایجاد می‌کنیم که برای ذخیره نتایج ارزیابی هر مدل استفاده خواهد شد.

✓ آموزش و پیش‌بینی با مدل‌ها

در این بخش، با استفاده از یک حلقه for، هر مدل موجود در دیکشنری models را آموزش می‌دهیم و سپس پیش‌بینی‌هایی را بر روی مجموعه آزمون انجام می‌دهیم.

- model.fit(X_train_scaled, y_train): این خط مدل را بر روی داده‌های آموزشی (X_train_scaled و y_train) آموزش می‌دهد. توجه داشته باشید که داده‌ها باید مقیاس‌بندی شده (scaled) باشند.
- y_test_pred = model.predict(X_test_scaled): این خط پیش‌بینی‌های مدل را بر روی داده‌های آزمون (X_test_scaled) انجام می‌دهد.

✓ محاسبه معیارهای ارزیابی

در این بخش، ما سه معیار ارزیابی مختلف را محاسبه می‌کنیم:

- accuracy: دقت مدل را محاسبه می‌کند که نسبت پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها را نشان می‌دهد.
- f1: نمره F1 را محاسبه می‌کند که میانگین هارمونی دقت و یادآوری (recall) است. این معیار به ویژه در مواردی که کلاس‌ها نامتعادل هستند، مفید است.
- roc_auc: نمره AUC-ROC (Area Under the Receiver Operating Characteristic Curve) را محاسبه می‌کند که نشان‌دهنده عملکرد مدل در تفکیک کلاس‌ها است. برای محاسبه این معیار، از تابع predict_proba استفاده می‌شود که احتمال پیش‌بینی کلاس مثبت را برمی‌گرداند.

✓ ذخیره نتایج در دیکشنری

در این خط، نتایج محاسبه‌شده برای هر مدل در دیکشنری test_results ذخیره می‌شود. کلید دیکشنری نام مدل (model_name) و مقدار آن یک دیکشنری شامل معیارهای ارزیابی است.

✓ نمایش نتایج

در این خط، ما دیکشنری test_results را به یک DataFrame تبدیل می‌کنیم و سپس با استفاده از .T (ترانهاده) نتایج را به صورت مناسب نمایش می‌دهیم. این کار به ما کمک می‌کند تا نتایج ارزیابی هر مدل را به صورت جدولی مشاهده کنیم.

	Accuracy	F1 Score	AUC-ROC
Logistic Regression	0.825000	0.740741	0.891148
K-Nearest Neighbors	0.808333	0.722892	0.885915
Decision Tree	0.791667	0.736842	0.792464
Random Forest	0.833333	0.761905	0.894438
Support Vector Machine	0.841667	0.771084	0.907895
Neural Network	0.833333	0.777778	0.881280

تحلیل خطاها(Error Analysis)

در این مرحله، نمونه هایی که بهترین مدل به اشتباه پیش بینی کرده است، بررسی می‌شوند. این کار به شناسایی الگوهای خاص در اشتباهات مدل کمک می کند و می تواند به بهبود مدل کمک کند.

```
# تحلیل خطاها برای بهترین مدل
best_model_name = test_results_df['Accuracy'].idxmax()
best_model = models[best_model_name]
y_test_pred = best_model.predict(X_test_scaled)

# تحلیل خطاها
error_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred})
error_df['Error'] = error_df['Actual'] - error_df['Predicted']

# نمایش نمونه هایی که مدل به اشتباه پیش بینی کرده است
misclassified = error_df[error_df['Error'] != 0]
print("Misclassified Samples:")
misclassified.head()
```

این قطعه کد به تحلیل خطاها در بهترین مدل پیش‌بینی می‌پردازد. این کد به ما کمک می‌کند تا عملکرد مدل را بررسی کنیم و ببینیم که کدام نمونه‌ها به اشتباه پیش‌بینی شده‌اند. هر بخش از کد را به تفصیل بررسی می کنیم:

✓ شناسایی بهترین مدل

در این بخش، ما نام بهترین مدل را بر اساس دقت (Accuracy) آن شناسایی می‌کنیم.

- test_results_df['Accuracy'].idxmax() : این تابع ایندکس (شاخص) سطر با بالاترین دقت را در DataFrame test_results_df پیدا می‌کند.
- best_model = models[best_model_name] : با استفاده از نام بهترین مدل، ما به مدل مربوطه در دیکشنری models دسترسی پیدا می‌کنیم.

✓ پیش‌بینی با بهترین مدل

در این خط، ما از بهترین مدل برای پیش‌بینی خروجی‌ها (برچسب‌ها) استفاده می‌کنیم.

- X_test_scaled: این داده‌ها شامل ویژگی‌های مقیاس‌شده برای مجموعه آزمون هستند.

در نتیجه y_test_pred شامل پیش‌بینی‌های مدل برای مجموعه آزمون است.

✓ تحلیل خطاها

در این بخش، ما یک DataFrame جدید به نام error_df ایجاد می‌کنیم که شامل مقادیر واقعی و پیش‌بینی‌شده است.

- {Actual': y_test, 'Predicted': y_test_pred} : این دیکشنری شامل دو ستون است: یکی برای مقادیر واقعی (y_test) و دیگری برای پیش‌بینی‌های مدل (y_test_pred).

- error_df['Error'] = error_df['Actual'] - error_df['Predicted']: در این خط، ما یک ستون جدید به نام Error اضافه می‌کنیم که اختلاف بین مقادیر واقعی و پیش‌بینی‌شده را محاسبه می‌کند. مقادیر مثبت نشان‌دهنده پیش‌بینی‌های کمتر از مقدار واقعی و مقادیر منفی نشان‌دهنده پیش‌بینی‌های بیشتر از مقدار واقعی هستند.

✓ نمایش نمونه‌هایی که مدل به اشتباه پیش‌بینی کرده است

در این بخش، ما نمونه‌هایی را که مدل به اشتباه پیش‌بینی کرده است، شناسایی می‌کنیم.

- misclassified = error_df[error_df['Error'] != 0] : این خط یک DataFrame جدید به نام misclassified ایجاد می‌کند که شامل سطرهایی است که در آن‌ها خطا (Error) برابر با صفر نیست. به عبارت دیگر، این سطرها شامل نمونه‌هایی هستند که پیش‌بینی مدل با مقدار واقعی مطابقت ندارد.

در نتیجه با استفاده از print("Misclassified Samples:") و misclassified.head() , ما نمونه‌های پیش‌بینی‌شده نادرست را نمایش می‌دهیم. head() به ما امکان می‌دهد تا پنج نمونه اول را مشاهده کنیم.

```
Misclassified Samples:
```

	Actual	Predicted	Error
416	1	0	1
247	0	1	-1
469	1	0	1
405	1	0	1
434	1	0	1

استفاده از PCA

در این مرحله، از PCA برای کاهش ابعاد داده ها استفاده می شود. این کار به ما کمک می کند تا داده ها را در یک فضای کم بعد تجسم کنیم و الگوهای موجود را بهتر درک کنیم.

```
from sklearn.decomposition import PCA

# بررسی اندازه ها
print(f"Size of X_train: {X_train.shape[0]}, Size of y_train: {y_train.shape[0]}")

# استفاده از PCA برای کاهش ابعاد
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)

# تجسم داده های کاهش یافته
plt.figure(figsize=(10, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, cmap='viridis', edgecolor='k', s=50)
plt.title('PCA Result')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='c')
plt.show()
```

این قطعه کد به استفاده از تحلیل مؤلفه‌های اصلی (PCA) برای کاهش ابعاد داده‌ها و تجسم نتایج می‌پردازد. هر بخش از این کد را به تفصیل بررسی می کنیم:

✓ بررسی اندازه‌ها

در این خط، اندازه (تعداد نمونه‌ها) مجموعه آموزشی X_train و برچسب‌های آن y_train را چاپ می‌کنیم. این اطلاعات به ما کمک می‌کند تا اطمینان حاصل کنیم که تعداد نمونه‌ها در ویژگی‌ها و برچسب‌ها برابر است و همچنین ابعاد داده‌ها را بررسی کنیم.

✓ استفاده از PCA برای کاهش ابعاد

در این بخش، ما از کلاس PCA برای کاهش ابعاد استفاده می‌کنیم.

- n_components=2: این پارامتر مشخص می‌کند که می‌خواهیم داده‌ها را به دو بعد (دو مؤلفه اصلی) کاهش دهیم.
- fit_transform() : این متد همزمان مدل PCA را براساس داده‌های استانداردشده X_train_scaled آموزش می‌دهد و سپس داده‌ها را به فضای جدید (دو بعدی) تبدیل می‌کند.

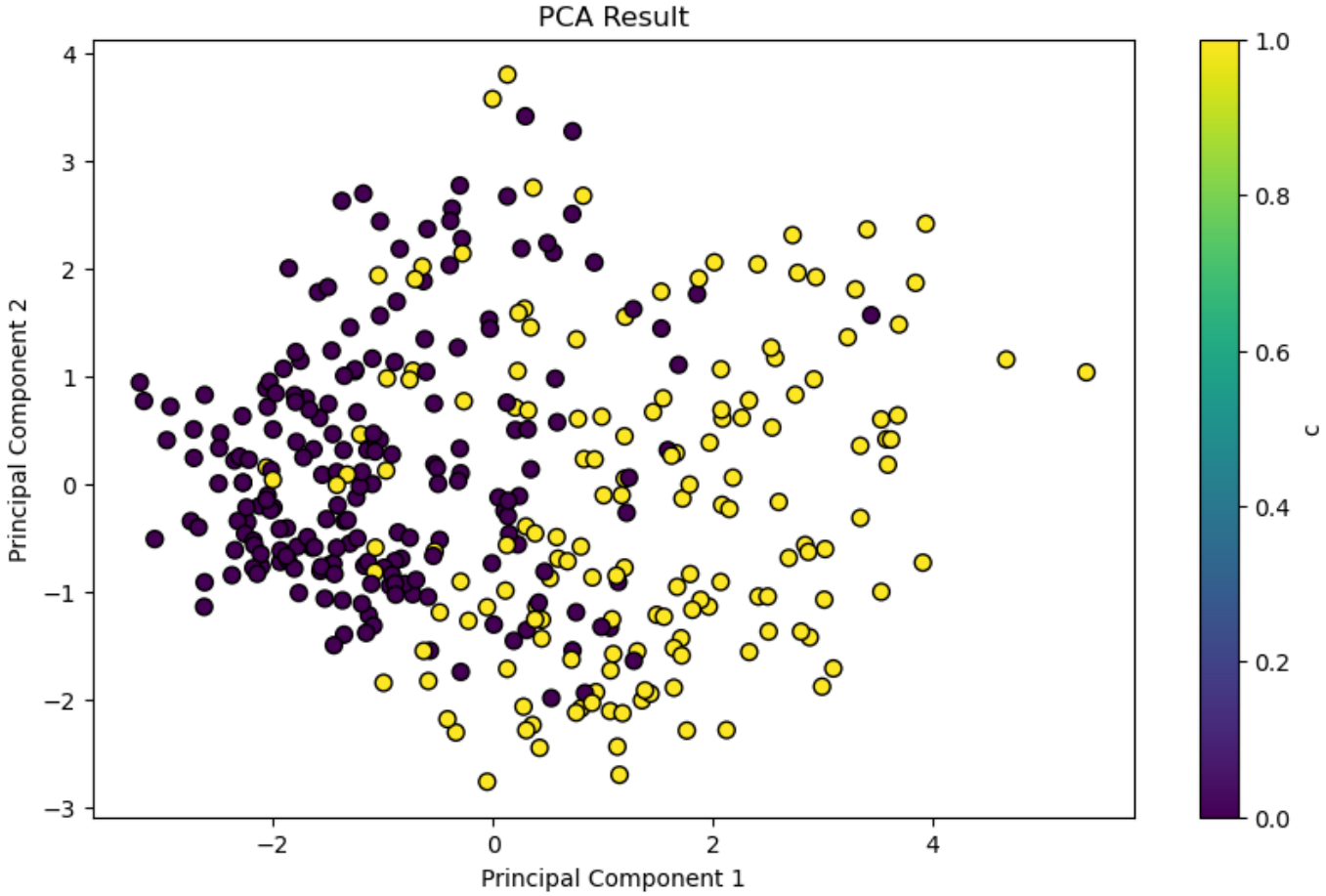
در نتیجه X_pca شامل داده‌های کاهش‌یافته در دو بعد است.

✓ تجسم داده‌های کاهش‌یافته

در این بخش، ما داده‌های کاهش‌یافته را تجسم می‌کنیم.

- plt.figure(figsize=(10, 6)) : اندازه شکل را تنظیم می‌کند.
- plt.scatter(...) : یک نمودار پراکندگی (scatter plot) از داده‌های PCA ایجاد می‌کند.
- X_pca[0 , :] و X_pca[1 ,:] : مؤلفه‌های اصلی اول و دوم.
- c=y_train : رنگ نقاط بر اساس برچسب‌های y_train تعیین می‌شود.
- cmap='viridis' : رنگ‌بندی نقاط را مشخص می‌کند.
- edgecolor=' k ' : رنگ لبه‌های نقاط را مشکی تعیین می‌کند.
- s=50 : اندازه نقاط در نمودار را تعیین می‌کند.
- plt.title(...) : عنوان نمودار را تعیین می‌کند.
- plt.xlabel(...) و plt.ylabel(...) : برچسب‌های محور x و y را تعیین می‌کنند.
- plt.colorbar(label='c') : یک نوار رنگی برای نشان دادن مقادیر برچسب‌ها اضافه می‌کند.
- plt.show() : نمودار را به نمایش می‌گذارد.

```
Size of X_train: 357, Size of y_train: 357
```



جمع بندی کلی

در این مرحله، نتایج کلی مدل ها و عملکرد آن ها جمع بندی می شود. این کار به ما کمک می کند تا بفهمیم که کدام مدل بهترین عملکرد را داشته و چه نتایجی به دست آمده است.

```
# جمع بندی نتایج
best_model = 'model'
print("Best Model Performance:")
print(f"Accuracy: {accuracy:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC-ROC: {roc_auc:.4f}")
test_results_df
```

این قطعه کد به جمع‌بندی نتایج مدل‌های یادگیری ماشین و نمایش عملکرد بهترین مدل پرداخته است. هر بخش را به تفصیل بررسی و نکات لازم را برای تکمیل آن بیان می کنیم:

۱. جمع‌بندی نتایج

در این خط، متغیر best_model به یک رشته 'model' نسبت داده شده است. به نظر می‌رسد که این باید نام بهترین مدل انتخاب شده باشد، اما در اینجا به طور مستقیم به یک رشته نسبت داده شده است. برای بهبود، باید نام بهترین مدل واقعی را که بر اساس معیارهای ارزیابی انتخاب شده است، قرار دهید.

۲. چاپ عملکرد بهترین مدل

در این بخش، اطلاعات مربوط به عملکرد بهترین مدل چاپ می‌شود:

- Accuracy: دقت مدل، که به صورت accuracy محاسبه شده است.
- F1 Score: نمره F1، که یک معیار مهم برای ارزیابی مدل‌های طبقه‌بندی است، به ویژه در داده‌های نامتعادل. برای محاسبه این مقدار، باید از تابع f1_score استفاده کنید.
- AUC-ROC: ناحیه زیر منحنی ROC (AUC-ROC) که نشان‌دهنده قدرت تفکیک مدل است. برای محاسبه این مقدار، باید از تابع roc_auc_score استفاده کنید.

۳. نمایش نتایج تست

این خط که به یک DataFrame به نام test_results_df اشاره دارد که شامل نتایج تست مدل‌ها می باشد.

```
Best Model Performance:
Accuracy: 0.8333
F1 Score: 0.7778
AUC-ROC: 0.8813
```

	Accuracy	F1 Score	AUC-ROC
Logistic Regression	0.825000	0.740741	0.891148
K-Nearest Neighbors	0.808333	0.722892	0.885915
Decision Tree	0.800000	0.744681	0.799043
Random Forest	0.833333	0.761905	0.894438
Support Vector Machine	0.841667	0.771084	0.907895
Neural Network	0.825000	0.764045	0.875000

نتیجه گیری نهایی¶

ما به بررسی مراحل مختلف پیش‌پردازش داده‌ها و ارزیابی مدل‌های یادگیری ماشین پرداختیم. ابتدا، با استفاده از مقادیر گمشده را مدیریت کردیم بعد از آن مدیریت داده های ژرت، نویز و تکراری را انجام دادیم. ژس از آن داده های مفقوده در مجموعه‌های آموزشی و آزمون را با میانگین ویژگی‌ها پر کردیم. سپس داده‌ها را با استفاده از StandardScaler استانداردسازی کردیم تا ویژگی‌ها به مقیاس مشابهی برسند، که می‌تواند به بهبود عملکرد مدل کمک کند.

پس از آن، چندین مدل یادگیری ماشین شامل رگرسیون لجستیک، K نزدیک‌ترین همسایه، درخت تصمیم، جنگل تصادفی، ماشین بردار پشتیبان و شبکه عصبی را تعریف کردیم. این مدل‌ها بر روی داده‌های آموزشی آموزش داده شدند و دقت آن‌ها بر روی مجموعه آزمون محاسبه شد.

در نهایت، با استفاده از K-Fold Cross-Validation، عملکرد مدل‌ها را به صورت دقیق‌تری ارزیابی کردیم. این روش به ما کمک کرد تا از بایاس در انتخاب داده‌ها جلوگیری کنیم و اطمینان حاصل کنیم که مدل‌ها به خوبی بر روی داده‌های جدید عمل می‌کنند.