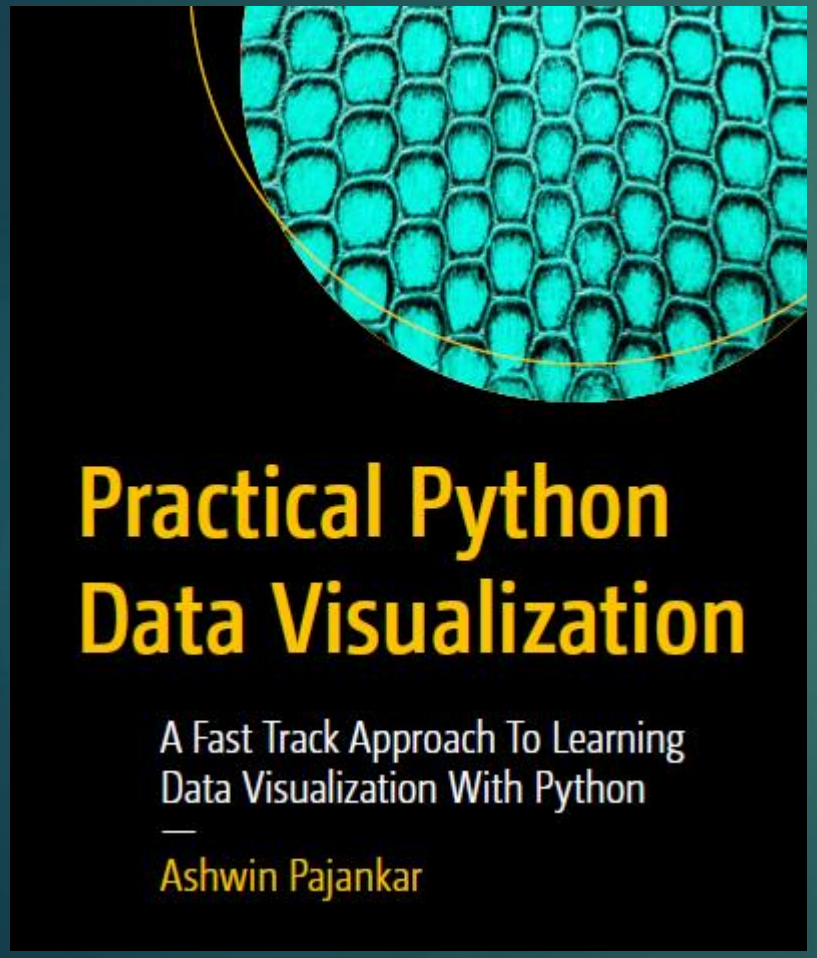
بنام خدا



پایتون کاربردی

تصویرسازی داده ها

یک رویکرد سریع برای یادگیری

تصویرسازی داده ها با پایتون

مباحث مورد نظر :

- فصل ۶: تصویرسازی تصاویر و اشکال سه بعدی
- دیدارسازی تصاویر
- عملیات روی تصاویر
- دیدارسازی های سه بعدی
- فصل ۷: تصویرسازی نمودارها و شبکه ها
- نمودارها و شبکه ها
- نمودارها در پایتون
- تصویرسازی نمودارها در پایتون
- انواع بیشتر نمودارها
- اختصاص برچسب های سفارشی به گره ها
- خلاصه

فصل ۶

تصویرسازی تصاویر و اشکال سه بعدی

مقدمه

در فصل ۵، با استفاده از کتابخانه Matplotlib در پایتون، دیدارسازی را شروع کردیم. در این فصل، ما ماجراجویی های خود را با Matplotlib و NumPy برای دیدارسازی تصاویر و اشکال سه بعدی ادامه خواهیم داد. بیایید کاوش خود را در مورد تصویرسازی داده ها با موضوعات زیر ادامه دهیم:

- دیدارسازی تصاویر
- عملیات روی تصاویر
- دیدارسازی سه بعدی

بعد از این فصل می توانید با تصاویر و اشکال سه بعدی کار کنید.

دیدارسازی تصاویر

در این بخش نحوه دیدارسازی تصاویر را یاد می گیریم. ما با ایجاد یک نوت بوک جدید برای این فصل شروع خواهیم کرد. در **Matplotlib** ،تابع **()imread** می تواند تصاویر را در قالب فرمت **png** بخواند. برای فعال کردن آن به خواندن فایل‌ها با فرمت‌های دیگر، باید کتابخانه پردازش تصویر دیگری به نام **pillow** را نصب کنید. با اجرای دستور زیر در سلول نوت بوک آن را نصب کنید:

```
!pip3 install pillow
```

از دستور جادویی که در فصل ۵ بحث کردیم استفاده کنید تا نوت بوک را برای نمایش تصاویر **Matplotlib** فعال کنید:

```
%matplotlib inline
```

ماژول **Pyplot Matplotlib** را با دستور زیر وارد کنید:

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

حالا بیایید با استفاده از تابع **()imread** یک تصویر را به صورت زیر در متغیر بخوانیم:

```
img1 = plt.imread('D:\\Dataset\\4.1.02.tiff')
```

اگر از رایانه لینوکس، یونیکس یا MacOS استفاده می کنید، باید از قرارداد زیر برای مسیر فایل استفاده کنید.

```
img1 = plt.imread('/home/pi/book/dataset/4.1.02.tiff’)
```

مسیری که از آن استفاده می کنید باید یک مسیر مطلق باشد. اگر فایل تصویر در همان مسیر برنامه باشد، فقط نام فایل کافی است. این تابع یک تصویر را می خواند و آن را در (NumPy ndarray) به عنوان ماتریس مقادیر عددی ذخیره می کند. با اجرای خط کد زیر می توانیم این موضوع را تأیید کنیم:

```
print(type(img1))
```

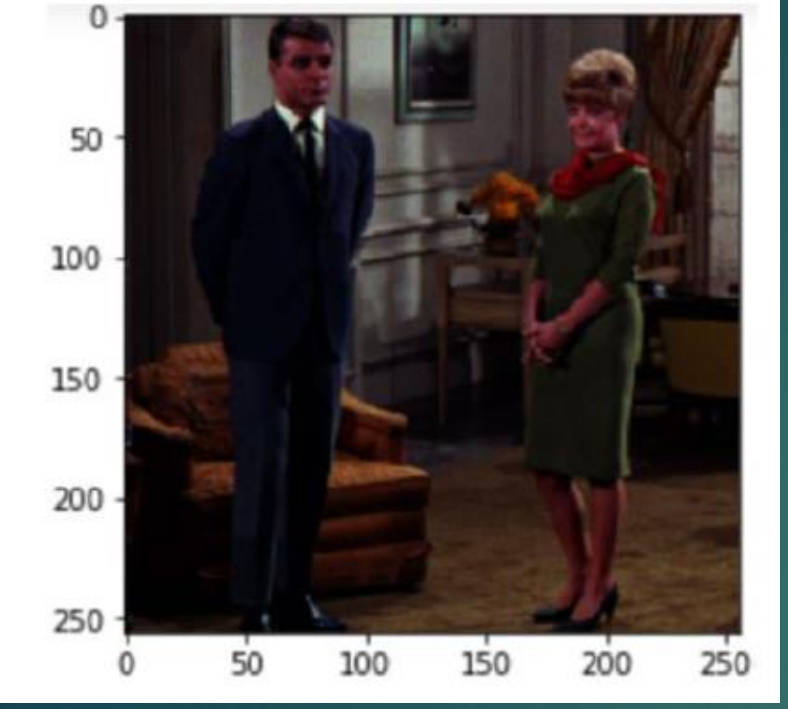
خروجی به صورت زیر است:

```
<class 'numpy.ndarray’>
```

این تایید می‌کند که کتابخانه‌های علمی پایتون، تصاویر را به‌عنوان درایه‌های NumPy مدیریت می‌کنند. می‌توانیم تصویر را با تابع imshow() از Matplotlib نشان دهیم و سپس با تابع show() به صورت شکل زیر نمایش دهیم:

```
plt.imshow(img1)
plt.show()
```

خروجی در شکل ۶-۱ نشان داده شده است.



شکل ۶-۱. دیدارسازی یک تصویر رنگی

می توانید محور را به صورت زیر خاموش کنید:

```
plt.imshow(img1)
plt.axis(False)
plt.show()
```

******توجه داشته باشید همه تصاویری که من برای نمایش استفاده می کنم از **http://www.imageprocessingplace.com/ root_files_V3/image_databases.htm.**

دانلود می شوند.

بیایید یک تصویر خاکستری را به صورت زیر بخوانیم:

```
img2 = plt.imread('D:\\Dataset\\5.3.01.tiff’)
plt.imshow(img2)
plt.axis(False)
plt.show()
```

شکل ۶-۲. دیدارسازی یک تصویر خاکستری با حالت پیش فرض نقشه رنگی

همانطور که می بینید رنگ ها کمی عجیب به نظر می رسند و این اصلا یک تصویر رنگی نیست. Matplotlib تصویر خاکستری را با نقشه رنگی پیش‌فرض نمایش می‌دهد. ما می توانیم نقشه رنگ خاکستری را به طور صریح برای دیدارسازی صحیح این تصویر به صورت زیر اختصاص دهیم:

```
plt.imshow(img2, cmap=plt.cm.gray)
plt.axis(False)
plt.show()
```

شکل ۶-۳. دیدارسازی یک تصویر در مقیاس خاکستری با نقشه رنگ خاکستری

همچنین می توانیم نقشه رنگی را به صورت زیر نشان دهیم:

```
plt.imshow(img2, cmap='gray')
plt.axis(False)
plt.show()
```

در ادامه به عملیات اصلی با تصاویر می پردازیم.

عملیات روی تصاویر

ما می‌توانیم با **NumPy** عملیات اولیه را روی تصاویر انجام دهیم و خروجی‌ها را با **Matplotlib** دیدارسازی کنیم. ابتدا چند عمل حسابی را یاد می گیریم. برای عملیات حسابی به دو تصویر با ابعاد یکسان نیاز داریم. اجازه دهید یک تصویر رنگی دیگر را به صورت زیر بخوانیم:

```
img3 = plt.imread('D:\\Dataset\\4.1.03.tiff’)
plt.imshow(img3)
plt.axis(False)
plt.show()
```

شکل ۶-۴. تصویر رنگی دیگر برای عملیات تصویر

می توانیم دو تصویر اضافه کنیم و آنها را به صورت زیر دیدارسازی کنیم:

```
add = img1+img3
plt.imshow(add)
plt.axis(False)
plt.show()
```

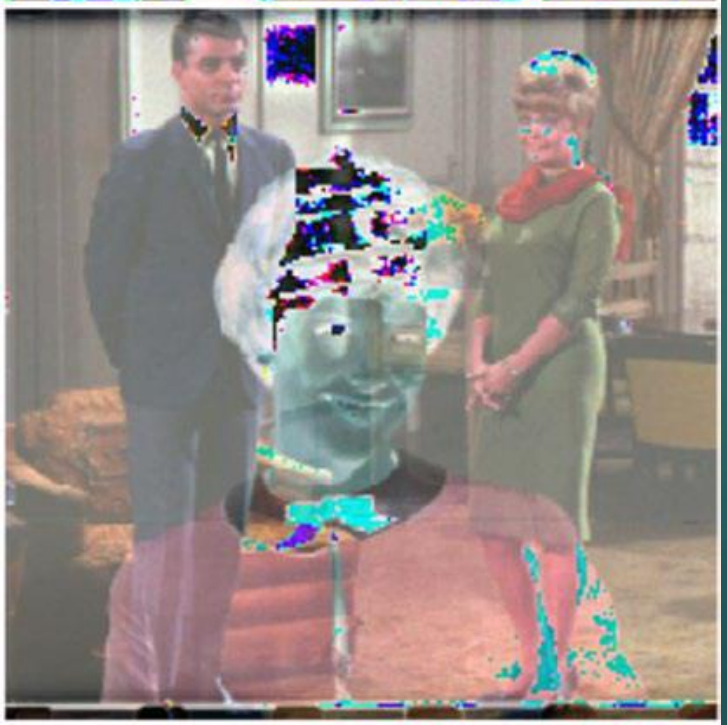
شکل ۶-۵. اضافه شدن دو تصویر

عمل جمع یک عمل حسابی جابجایی است. یعنی اگر ترتیب عملوندها را تغییر دهیم روی خروجی تاثیری نخواهد داشت.

```
add1 = img3+img1
plt.imshow(add)
plt.axis(False)
plt.show()
```

بیایید عملیات تفریق را امتحان کنیم:

```
sub1= img1-img3
plt.imshow(sub1)
plt.axis(False)
plt.show()
```



شکل ۶-۶. نتیجه تفریق

می دانیم که عمل تفریق جابجایی نیست. یعنی اگر ترتیب عملوندها را تغییر دهیم، نتیجه متفاوت است. بیایید امتحان کنیم که:

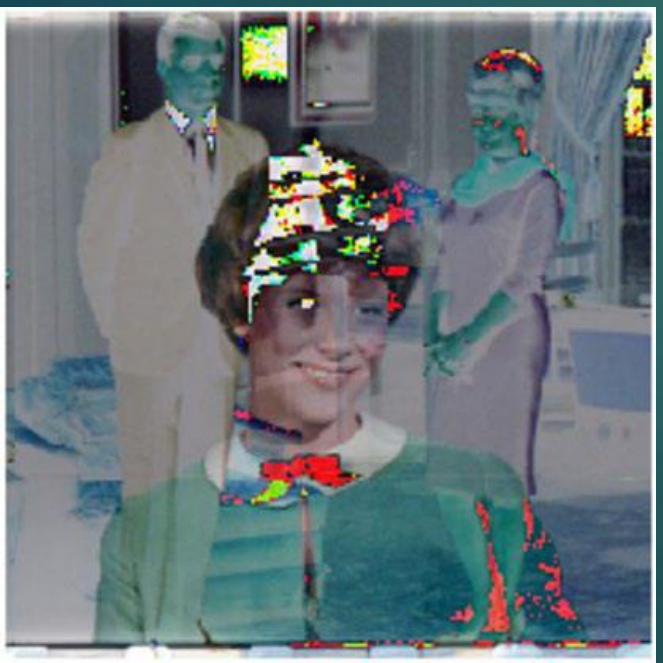
```
sub1 = img3-img1

plt.imshow(sub1)

plt.axis(False)

plt.show()
```

بنابراین خروجی نشان داده شده در شکل ۶-۷ با خروجی قبلی متفاوت است (شکل ۶-۶).



شکل ۶-۷. نتیجه تفریق

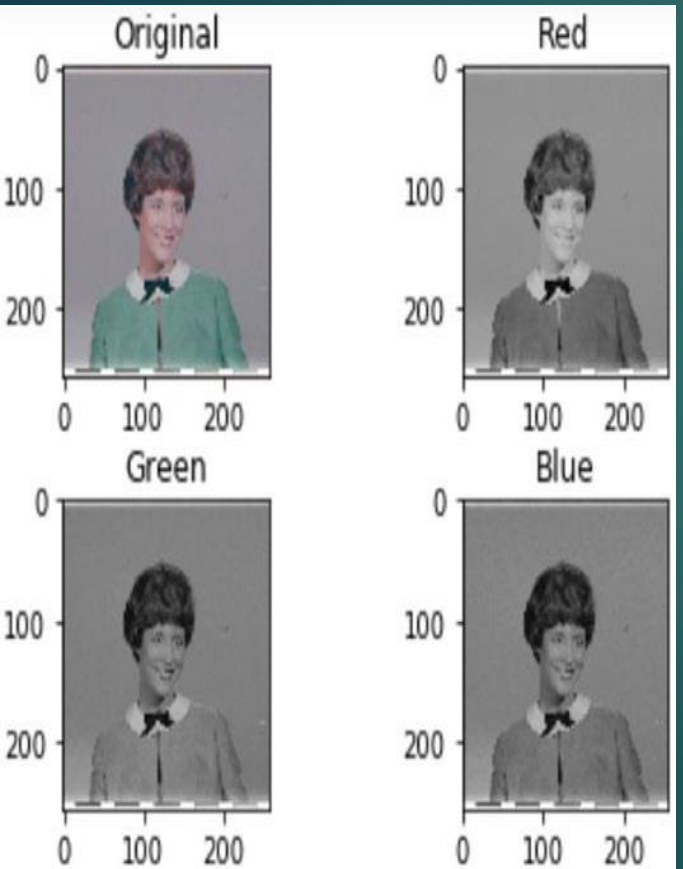
ما آموخته ایم که تصاویر در SciPy به صورت NumPy ndarray نمایش داده می شوند. تفاوت بین تصاویر رنگی و خاکستری در این است که تصاویر رنگی از چندین کانال تشکیل شده اند. این کانال ها ، خود دارنده ها هستند. می‌توانیم تصاویر رنگی را تقسیم کنیم و کانال‌های تشکیل‌دهنده را با استفاده از نمایه‌سازی در NumPy مجزا نماییم. تصاویر رنگی برای هر یک از رنگ های قرمز، سبز و آبی یک کانال دارند. می توانید یک تصویر را با نمایه سازی NumPy به صورت زیر تقسیم کنید:

```
r = img3[:, :, 0]
g = img3[:, :, 1]
b = img3[:, :, 2]
```

بیایید تصویر اصلی و کانال های رنگی را با استفاده از نمودارهای فرعی در Matplotlib تصویرسازی کنیم:

```
plt.subplots_adjust(hspace=0.4, wspace=0.1)
plt.subplot(2, 2, 1)
plt.title('Original’)
plt.imshow(img3)
plt.subplot(2, 2, 2)
plt.title('Red')
plt.imshow(r, cmap='gray')
plt.subplot(2, 2, 3)
plt.title('Green')
plt.imshow(g, cmap='gray')
plt.subplot(2, 2, 4)
plt.title('Blue')
plt.imshow(b, cmap='gray')
plt.show()
```

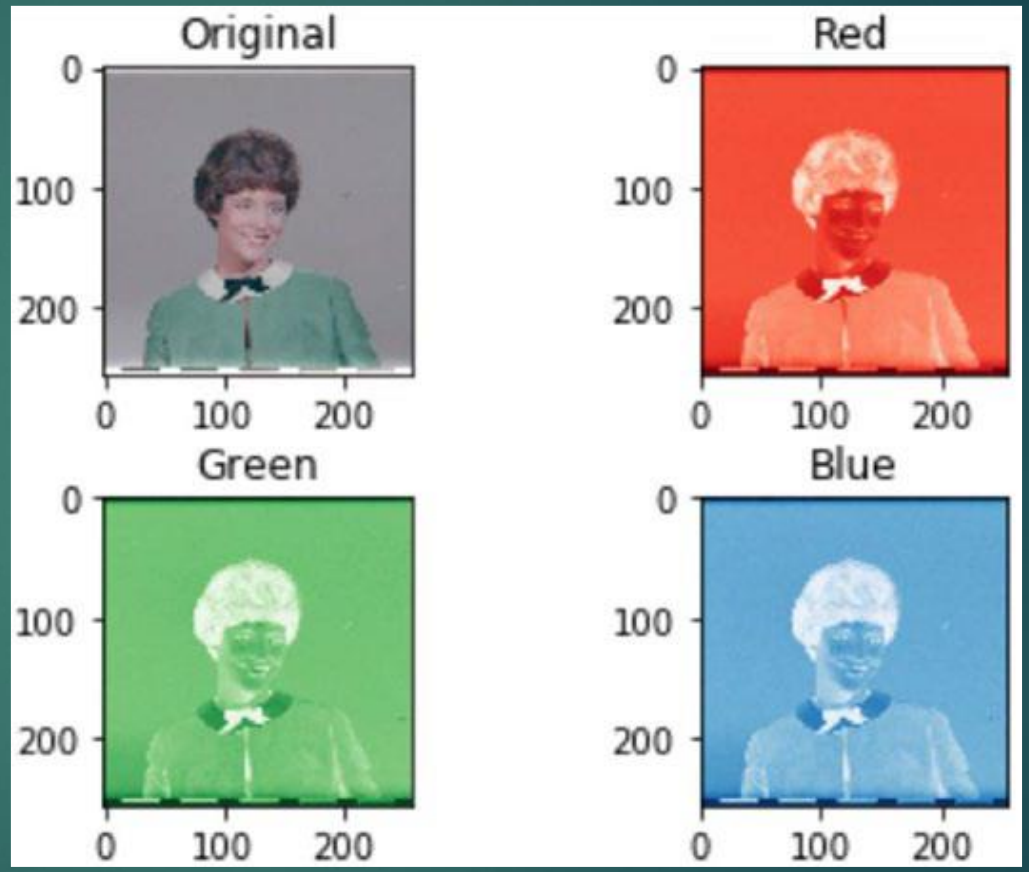
بیایید کد قبلی را بررسی کنیم. شما با استفاده از تابع subplot() یک شبکه ۲×۲ ایجاد می کنید. تابع () subplots_adjust برای تنظیم فاصله بین تصویر ها استفاده می شود. موقعیت بالا سمت چپ در شبکه ۲ × ۲ ،۱ است، موقعیت مجاور در همان ردیف ۲ است و غیره. موقعیت پایین سمت راست، موقعیت چهارم است. با استفاده از تابع subplot() قبل از imshow() یا plot() ، می توانیم تصمیم بگیریم که تصویر در کجا قرار گیرد. خروجی در شکل ۶-۸ نشان داده شده است.



شکل ۶-۸. کانال های رنگی جدا شده

کانال‌های رنگی خود ماتریس‌های دوبعدی هستند که مقادیر اعضای آن از ۰ تا ۲۵۶ برای قالب نمایش عدد صحیح بدون علامت ۸ بیتی یک تصویر رنگی متغیر است. هنگامی که آنها ترکیب می شوند، Matplotlib آنها را به عنوان یک تصویر رنگی تفسیر می کند. برای تصویرسازی این موضوع می توانیم از نقشه های رنگی مناسب به صورت زیر استفاده کنیم:

```
plt.subplots_adjust(hspace=0.4, wspace=0.1)
plt.subplot(2, 2, 1)
plt.title('Original')
plt.imshow(img3)
plt.subplot(2, 2, 2)
plt.title('Red')
plt.imshow(r, cmap='Reds')
plt.subplot(2, 2, 3)
plt.title('Green’)
plt.imshow(g, cmap='Greens')
plt.subplot(2, 2, 4)
plt.title('Blue')
plt.imshow(b, cmap='Blues')
plt.show()
```



شکل ۶-۹. کانال های رنگی جدا شده با نقشه های رنگی مناسب تصویرسازی شده اند

می‌توانیم کانال‌های سازنده را با هم ترکیب کنیم تا تصویر اصلی را با استفاده از کد زیر تشکیل دهیم:

```
import numpy as np
img4 = np.dstack((r, g, b))
```

ما از تابع **dstack()** از کتابخانه **NumPy** استفاده می کنیم. می توانیم خروجی را با کد معمولی تصویرسازی کنیم:

```
plt.imshow(img4)
plt.axis(False)
plt.show()
```

به این ترتیب می توانید عملیات بسیار ابتدایی پردازش تصویر را انجام دهید. پردازش تصویر بیش از آنچه آموخته ایم وجود دارد، و نمایش های بیشتر نیاز به کتاب جداگانه ای دارد.

دیدارسازی های سه بعدی

در این قسمت به بررسی تصویرسازی های سه بعدی با پایتون می پردازیم. تا این مرحله، ما دیدارسازی‌ها را در خود نوت بوک نمایش می‌دهیم. این فرآیند برای تصویرسازی های دو بعدی به خوبی کار می کند. با این حال، در مورد تصویرسازی های سه بعدی، آنها را فقط با یک زاویه ثابت نشان می دهد. بنابراین باید از روش دیگری استفاده کنیم. بهترین راه برای کار با این، استفاده از دستور جادویی دیگری است که تصویرسازی را در یک پنجره Qt جداگانه به صورت زیر نشان می دهد:

```
%matplotlib qt
```

ما قبلاً ماژول pyplot را در Matplotlib و NumPy در سلول‌های قبلی وارد کرده‌ایم (من فرض می‌کنم که از همان نوت بوک برای کل فصل استفاده خواهید کرد). ما باید عملکرد بیشتری را با عبارات زیر وارد کنیم:

```
from mpl_toolkits.mplot3d import Axes3D
```

```
from mpl_toolkits.mplot3d import axes3d
```

بیایید با یک چیز ساده شروع کنیم. ما نحوه ترسیم نمودارهای ساده را دیدیم، بنابراین اکنون همین کار را در سه بعدی انجام خواهیم داد. با منحنی یک پارامتر ساده شروع کنید. ابتدا شکل و محور را تعریف کنید:

```
fig = plt.figure()
```

```
ax = fig.gca(projection='3d')
```

برای تصویرسازی های سه بعدی، پیش بینی های سه بعدی را با پارامتر و جفت آرگومان projection='3d' فعال کرده ایم. بعد، داده ها را تعریف کنید. شما به مختصات x، yو z نقاط داده نیاز دارید. ابتدا مختصات قطبی نقاط داده را به صورت زیر تعریف کنید:

```
theta = np.linspace(-3 * np.pi, 3 * np.pi, 200)

z = np.linspace(-3, 3, 200)

r = z**3 + 1
```

اکنون مختصات xو y را به صورت زیر محاسبه کنید:

```
x = r * np.sin(theta)

y = r * np.cos(theta)
```

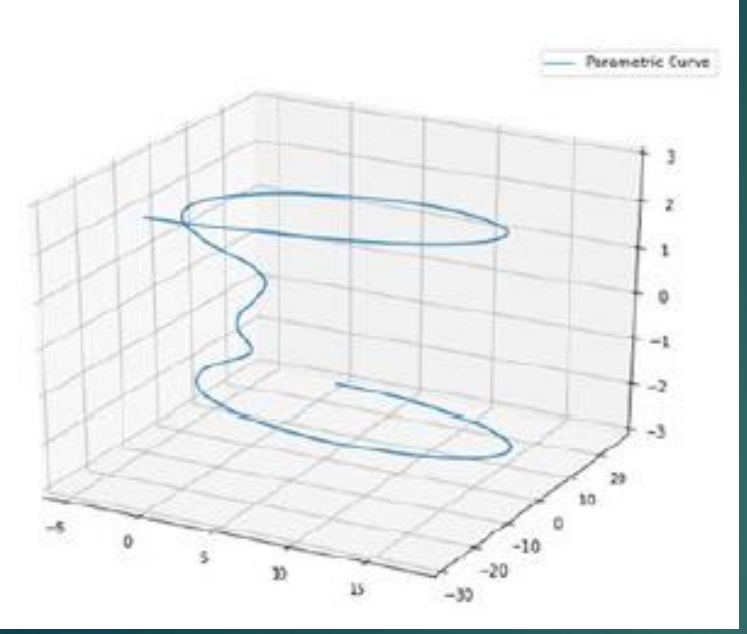
ما از توابع مثلثاتی در کتابخانه NumPy برای محاسبه x و y استفاده می کنیم. در نهایت بیایید آن را به صورت زیر ترسیم کنیم:

```
ax.plot(x, y, z, label='Parametric Curve')

ax.legend()

plt.show()
```

خروجی همانطور که در شکل ۶-۱۰ نشان داده شده است در یک پنجره جداگانه نمایش داده می شود و به گونه ای تعاملی خواهد بود که می توانیم زاویه دید را تغییر دهیم.



شکل ۶-۱۰. تصویرسازی یک منحنی پارامتریک

*****توجه داشته باشید اگر یک پنجره خروجی جداگانه باز نشود و تصویرسازی سه بعدی در خود مرورگر نشان داده شود، سپس باید هسته Jupyter را مجددا راه اندازی کنید و سلول های مربوطه را دوباره اجرا کنید. در این صورت، سلول حاوی %matplotlib درون خطی را اجرا نکنید.

در مرحله بعد، بیایید یک نمودار میله ای سه بعدی ایجاد کنیم. اکنون داده ها را به صورت زیر محاسبه کنید:

```
x = np.arange(4)
y = np.arange(4)
xx, yy = np.meshgrid(x, y)
print(xx);print(yy)
```

تابع meshgrid() ماتریس های مختصات را از بردارهای مختصات به صورت زیر ایجاد و برمی گرداند:

```
[[0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]]
[[0 0 0 0]
 [1 1 1 1]
 [2 2 2 2]
 [3 3 3 3]]
```

سپس از تابع ravel() برای صاف کردن هر دو ماتریس به صورت زیر استفاده کنید:

```
X, Y = xx.ravel(), yy.ravel()
print(X); print(Y);
```

این منجر به خروجی زیر می شود:

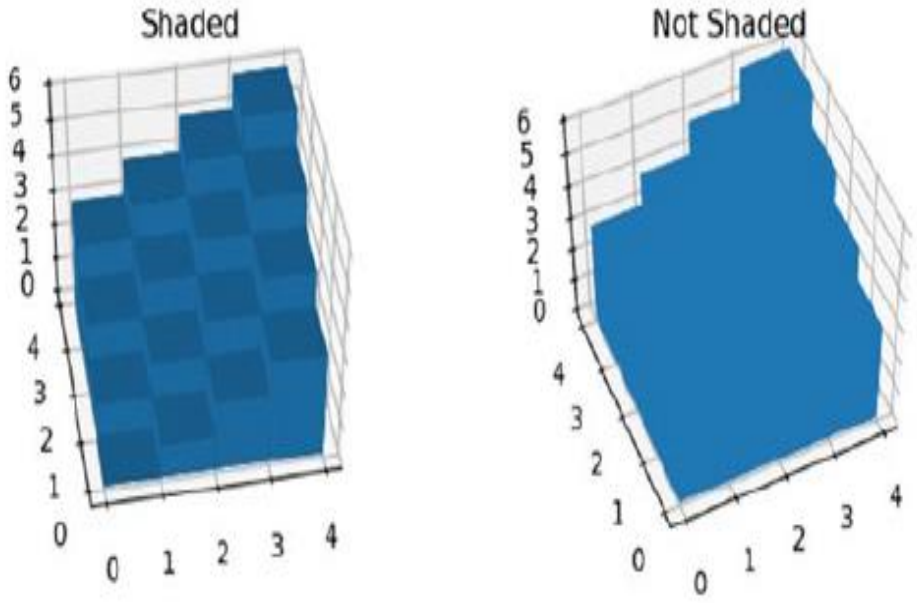
```
[0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]
[0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3]
```

اکنون مقادیر بیشتری را محاسبه کنید:

```
top = X + Y
bottom = np.zeros_like(top)
width = depth = 1
```

سپس یک شکل و محور ایجاد کنید و نمودار میله ای را به صورت زیر ایجاد کنید:

```
fig = plt.figure(figsize=(8, 3))
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122, projection='3d')
ax1.bar3d(X, Y, bottom, width, depth, top, shade=True)
ax1.set_title('Shaded')
ax2.bar3d(X, Y, bottom, width, depth, top, shade=False)
ax2.set_title('Not Shaded')
plt.show()
```

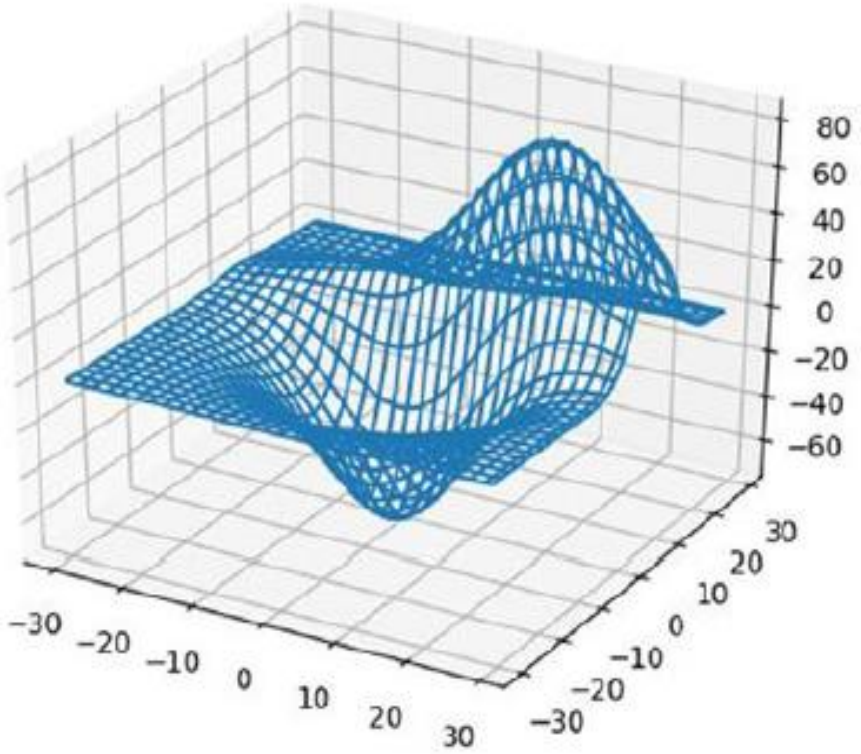


شکل ۶-۱۱. تصویرسازی یک نمودار میله ای سه بعدی

بعد بیایید یک **Wireframe** را به صورت زیر تصویرسازی کنیم:

```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y, Z = axes3d.get_test_data(delta=0.1)
ax.plot_wireframe(X, Y, Z)
plt.show()
```

در این قطعه کد ابتدا یک شکل و یک محور در حالت سه بعدی ایجاد می کنیم. سپس تابع داخلی **get_test_data()** داده های تست را برمی گرداند. سپس از تابع **plot_wireframe()** برای رسم مدل **wireframe** استفاده می کنیم. خروجی در شکل ۶-۱۲ نشان داده شده است.



شکل ۶-۱۲. تصویرسازی یک قاب سیمی سه بعدی(رویه)

در نهایت، بیایید یک سطح را محاسبه و تصویرسازی کنیم. ابتدا مختصات x و y را محاسبه کنید و سپس شبکه مش را محاسبه کنید:

```
x = np.arange(-3, 3, 0.09)
y = np.arange(-3, 3, 0.09)
X, Y = np.meshgrid(x, y)
print(X); print(Y)
```

خروجی به صورت زیر است:

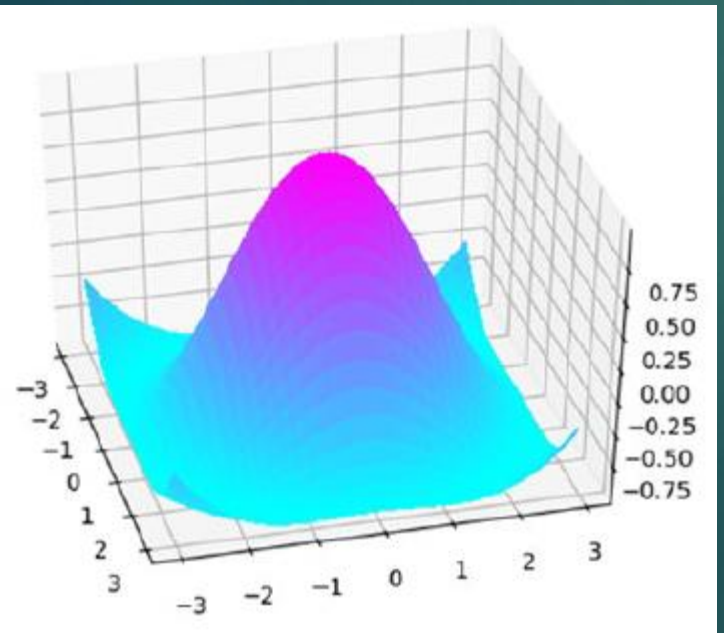
```
[[-3.   -2.91 -2.82 ...  2.76  2.85  2.94]
 [-3.   -2.91 -2.82 ...  2.76  2.85  2.94]
 [-3.   -2.91 -2.82 ...  2.76  2.85  2.94]
 ...
 [-3.   -2.91 -2.82 ...  2.76  2.85  2.94]
 [-3.   -2.91 -2.82 ...  2.76  2.85  2.94]
 [-3.   -2.91 -2.82 ...  2.76  2.85  2.94]]
[[-3.   -3.   -3.   ... -3.   -3.   -3.  ]
 [-2.91 -2.91 -2.91 ... -2.91 -2.91 -2.91]
 [-2.82 -2.82 -2.82 ... -2.82 -2.82 -2.82]
 ...
 [ 2.76  2.76  2.76 ...  2.76  2.76  2.76]
 [ 2.85  2.85  2.85 ...  2.85  2.85  2.85]
 [ 2.94  2.94  2.94 ...  2.94  2.94  2.94]]
```

سپس مختصات z را به صورت زیر محاسبه کنید:

```
R = np.sqrt(X**2 + Y**2)
Z = np.cos(R)
print(Z)
```

در نهایت شکل و محور را ایجاد کنید و سطح سه بعدی را تصویرسازی کنید:

```
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z,
                       cmap=plt.cm.cool,
                       linewidth=0,
                       antialiased=False)
plt.show()
```



شکل ۶-۱۳. تصویرسازی یک سطح سه بعدی

خلاصه

در این فصل، شما با دیدارسازی تصاویر آشنا شدید و آنها را نشان دادید. شما همچنین در مورد عملیات های اساسی روی تصاویر مانند عملیات حسابی و تقسیم تصاویر رنگی به کانال های تشکیل دهنده اطلاعات نشان دادید. ما همچنین برنامه‌های نوشتن برای دیدارسازی های سه بعدی شامل منحنی‌ها، میله‌ها، قاب‌های سیمی و مش‌ها را پوشش دادیم. فصل بعدی چگونگی دیدارسازی شبکه ها و ساختارهای داده گراف را بررسی می کند.

فصل ۷

تصویرسازی نمودارها و شبکه ها

مقدمه

در فصل ۶، دیدارسازی تصاویر و اشیاء سه بعدی را با پایتون و Matplotlib نشان دادیم. ما همچنین کمی پردازش تصویر را یاد گرفتیم. این فصل بر روی دیدارسازی ساختار داده معروف به گراف یا شبکه تمرکز دارد. بیایید سفر تصویرسازی داده خود را با موضوعات زیر ادامه دهیم:

- نمودارها و شبکه ها
- تصویرسازی نمودارها در پایتون
- انواع بیشتر نمودارها
- اختصاص برچسب های سفارشی به گره ها

پس از تکمیل این فصل، می‌توانید نمودارها و شبکه‌ها را با پایتون تصویرسازی کنید.

نمودارها و شبکه ها

گراف یک نوع داده انتزاعی است و به عنوان شبکه نیز شناخته می شود. این شامل مجموعه محدودی از رئوس (همچنین به عنوان گره) و مجموعه محدودی از یال ها (همچنین به عنوان پیوند شناخته می شود) است. اصطلاح رئوس و یال زمانی که به یک نمودار اشاره می کنیم استفاده می شود. گره ها و پیوندها اصطلاحاتی هستند که وقتی به همان ساختار شبکه اشاره می کنیم استفاده می شود. در سرتاسر فصل، من از اصطلاحات نمودارها، گره‌ها و یال‌ها برای ارجاع به این ساختار داده برای سازگاری استفاده می‌کنم. در یک نمودار، راس ها توسط یال ها به یکدیگر متصل می شوند. درختان زیرگروه ها هستند. در نمودارها می توانیم چرخه(دور) داشته باشیم، اما در درختان نمی توانیم. ما همچنین می توانیم نمودارهای جهت دار و گراف های غیر جهت دار داشته باشیم. یال ها را می توان با مقادیری در نمودارها نسبت داد که این نمودارها ، به عنوان نمودارهای وزنی شناخته می شوند. در این فصل، ما در مورد تصویرسازی نمودارهای بدون جهت و بدون وزن بحث می کنیم.

نمودارها در پایتون

برای کار و تصویرسازی با نمودارها، یک کتابخانه با کاربری آسان برای پایتون وجود دارد که به نام **networkx** شناخته می شود. می توانید با اجرای دستور زیر در سلول **Jupyter Notebook** آن را نصب کنید:

```
!pip3 install network
```

اکنون آن و کتابخانه دیگر **Matplotlib** را وارد کنید. ما همچنین رسم نمودار را در نوت بوک فعال خواهیم کرد:

```
%matplotlib inline
import networkx as nx
import matplotlib.pyplot as plt
```

می توانید یک نمودار خالی جدید به صورت زیر ایجاد کنید:

```
G = nx.Graph()
```

اجازه دهید نوع آن را به صورت زیر تعیین کنیم:

```
type(G)
```

در اینجا خروجی بصورت زیر است:

```
networkx.classes.graph.Graph
```

شما می توانید لیست گره ها و یال ها را ببینید و همچنین می توانید نوع داده آنها را به صورت زیر مشاهده کنید:

```
print(G.nodes())
print(G.edges())
print(type(G.nodes()))
print(type(G.edges()))
```

این خروجی زیر را تولید می کند:

```
[]
[]
<class 'networkx.classes.reportviews.NodeView'>
<class 'networkx.classes.reportviews.EdgeView'>
```

می توانید یک گره به صورت زیر اضافه کنید:

```
G.add_node('a’)
```

از طرف دیگر، می توانید چندین گره مشخص شده در یک لیست را نیز اضافه کنید:

```
G.add_nodes_from(['b', 'c'])
```

ما دوباره لیست گره ها و لیست یال ها را چاپ می کنیم (که باید خالی باشد، زیرا هنوز هیچ یالی اضافه نکرده ایم):

```
print('Nodes of the graph G: ')
print(G.nodes())
print('Edges of the graph G: ')
print(G.edges())
```

خروجی به صورت زیر است:

```
Nodes of the graph G:
['a', 'b', 'c']
Edges of the graph G:
[]
```

به عبارت زیر توجه کنید:

```
G.add_edge(1, 2)
```

دو گره و یک یال مربوطه اضافه می کند. اگر آرگومان در فراخوانی تابع قبلاً بخشی از لیست گره‌های گراف باشد، دو بار اضافه نمی‌شود. راه دیگری برای اضافه کردن آن در زیر آمده است:

```
edge = ('d', 'e')
G.add_edge(*edge)
edge = ('a', 'b')
G.add_edge(*edge)
```

بیایید دوباره لیست گره ها و یال ها را به صورت زیر چاپ کنیم:

```
print('Nodes of the graph G: ')
print(G.nodes())
print('Edges of the graph G: ')
print(G.edges())
```

همچنین می توانیم لیست یال ها را مشخص کرده و به نمودار اضافه کنیم:

```
G.add_edges_from([('a', 'c'), ('c', 'd’),
                  ('a', 1), (1, 'd’),
                  ('a', 2)])
```

حالا بیایید دوباره همه چیز را چاپ کنیم:

```
print('Nodes of the graph G: ')
print(G.nodes())
print('Edges of the graph G: ')
print(G.edges())
```

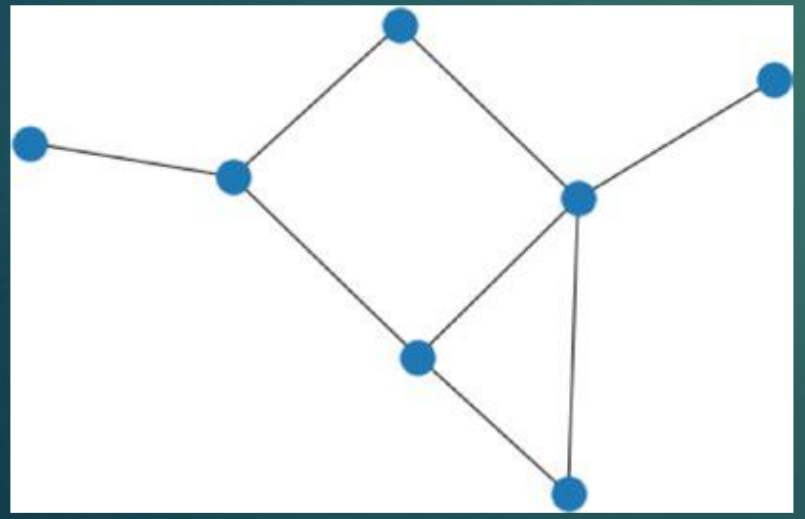
خروجی به صورت زیر است:

```
Nodes of the graph G:
['a', 'b', 'c', 1, 2, 'd', 'e']
Edges of the graph G:
[('a', 'b'), ('a', 'c'), ('a', 1), ('a', 2), ('c', 'd'), (1,2), (1, 'd'), ('d', 'e')]
```

تصویرسازی گراف ها در پایتون

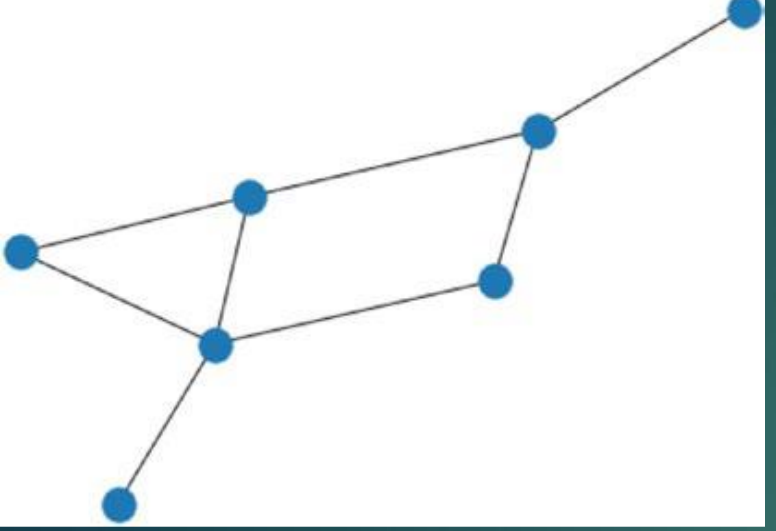
این بخش بر روی تصویرسازی نمودارها با کتابخانه **networkx** تمرکز دارد. شما قبلا یک نمودار را در بخش قبلی آماده کرده اید. اکنون می توانید آن را به صورت زیر تصویرسازی کنید:

```
nx.draw(G)
plt.show()
```



شکل ۷-۱. نمایش تصویری نمودار

توجه داشته باشید که این تصویرسازی‌ها به صورت تصادفی تولید می شوند و هر بار که عبارت را اجرا می کنیم، تصویر متفاوتی برای نمایش دیداری ایجاد می کند. با این حال، نمودارهای ارائه شده توسط این تصویرسازی‌ها، هم شکل هستند.



شکل ۷-۲. یک نمایش دیداری دیگر از همان نمودار

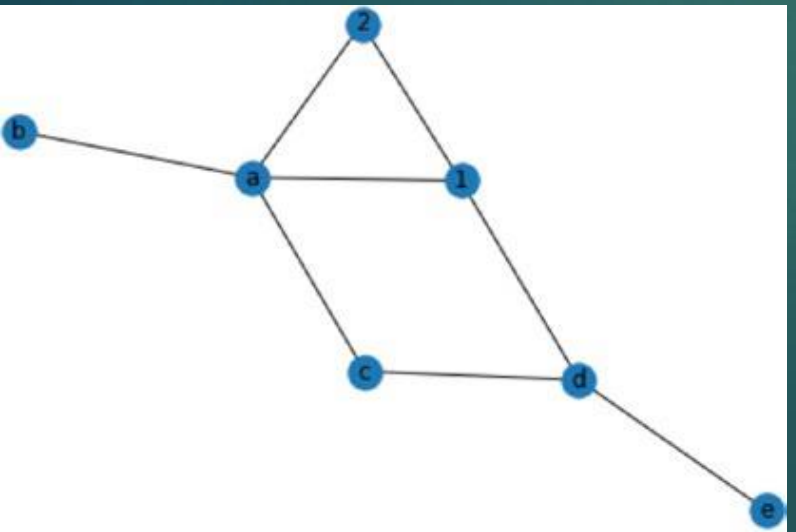
می توانید ببینید که هر دو تصویرسازی هم شکل هستند، زیرا از یک نمودار تولید می شوند.

همچنین باید مشاهده کرده باشید که گره ها هیچ نامی را نمایش نمی دهند.

با کد زیر می توانید نام گره ها را نشان دهید:

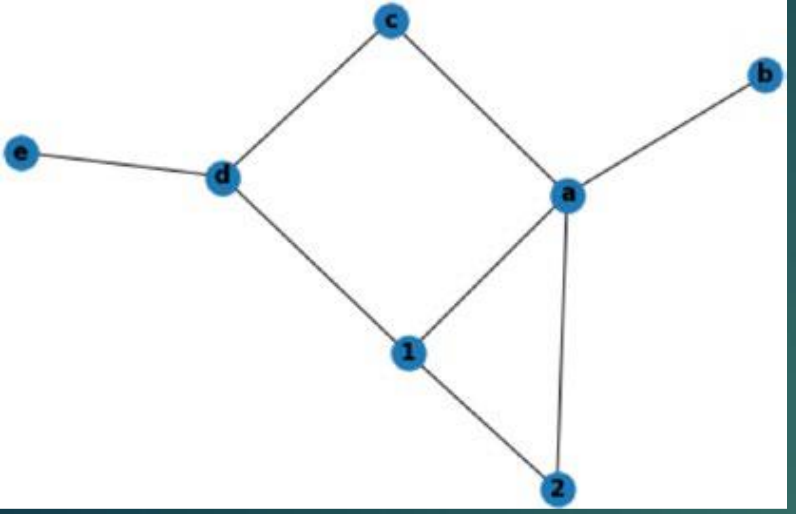
```
nx.draw(G, with_labels=True)
plt.show()
```

شکل ۷-۳. گره های دارای برچسب



برای نمایش برچسب گره ها به صورت پررنگ از این کد استفاده کنید:

```
nx.draw(G, with_labels=True, font_weight='bold’)
plt.show()
```



شکل ۷-۴. گره هایی با برچسب های پررنگ

انواع بیشتر نمودارها

در مرحله بعد، می توانیم به چند نوع دیگر از نمودارها و تصویرسازی آنها نگاه کنیم. اولین گراف خطی یا گراف مسیر است. تعداد گره های متصل به یک گره در یک گراف به عنوان درجه آن گره شناخته می شود. می توان گفت که گراف مسیر، گرافی است که دارای دو گره درجه یک و بقیه گره ها دارای درجه دو هستند. اگر یک نمودار مسیر را به صورت دیداری مرتب کنیم، شبیه یک خط قطعه قطعه شده است. برای ایجاد نمودار مسیر، کد زیر را اجرا کنید:

```
G = nx.path_graph(4)
```

برای نمایش گره ها و یال ها از این کد استفاده کنید:

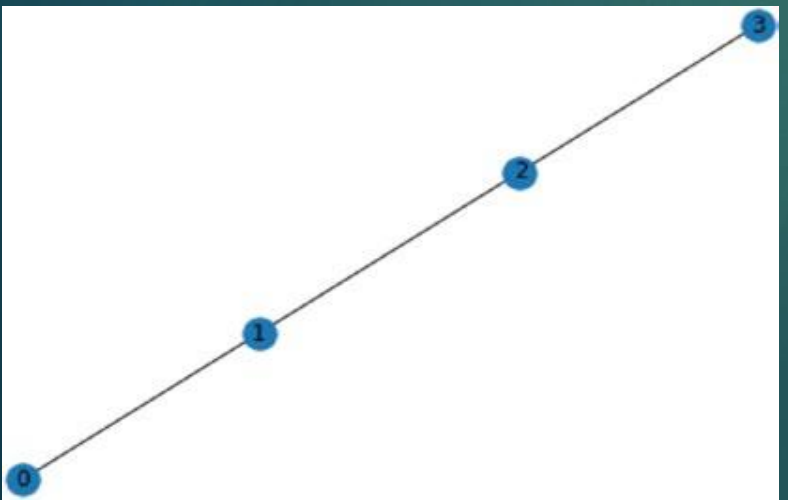
```
print('Nodes of the graph G: ')
print(G.nodes())
print('Edges of the graph G: ')
print(G.edges())
```

در اینجا خروجی بصورت زیر است:

```
Nodes of the graph G:
[0, 1, 2, 3]
Edges of the graph G:
[(0,1), (1,2), (2,3)]
```

بعد، بیایید تصویری از آن ایجاد کنیم:

```
nx.draw(G, with_labels=True)
plt.show()
```



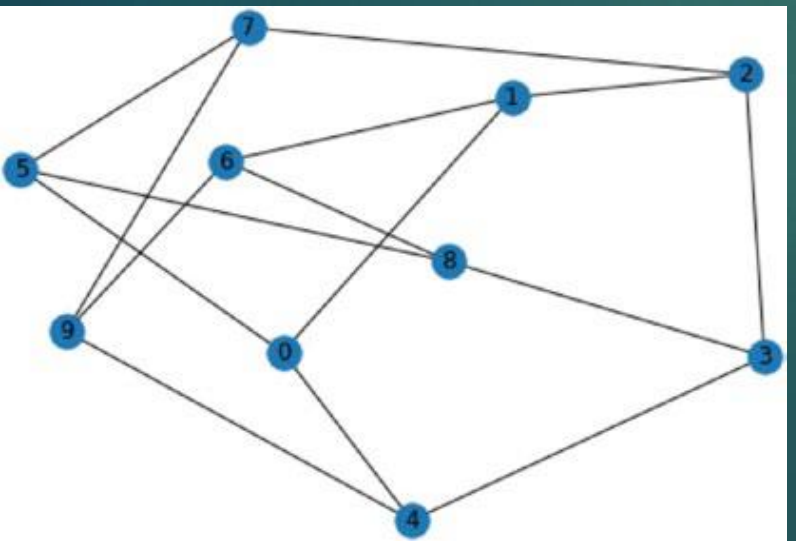
شکل ۷-۵. نمودار مسیر با چهار گره

نوع بعدی گراف، گراف پترسن است، یک گراف بدون جهت با ۱۰ راس و ۱۵ یال. ما می توانیم آن را ایجاد کنیم و مقادیر یال ها و گره ها را با استفاده از کد زیر مشاهده کنیم:

```
G = nx.petersen_graph()
print('Nodes of the graph G: ')
print(G.nodes())
print('Edges of the graph G: ')
print(G.edges())
```

می توانیم آن را به صورت زیر تصویرسازی کنیم:

```
nx.draw(G, with_labels=True)
plt.show()
```

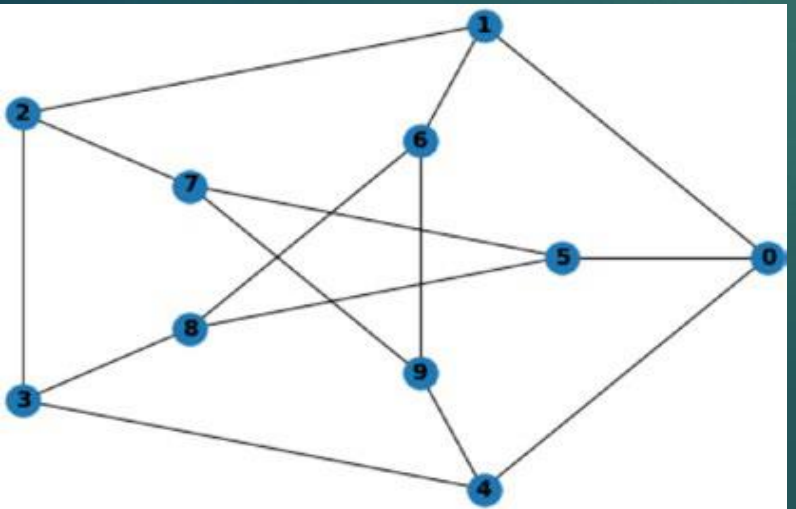


شکل ۷-۶. نمودار پترسن

در حال حاضر، راه های زیادی برای به تصویر کشیدن بهتر آن وجود دارد. می‌توانیم آن را به صورت یک پنج ضلعی با یک پنتاگرام در داخل، با پنج پره (پنج ضلعی) به تصویر بکشیم. کد زیر این کار را انجام می دهد.

```
nx.draw_shell(G, nlist=[range(5, 10),
                        range(5)],
              with_labels=True,
              font_weight='bold’)
plt.show()
```

این کد آن را به صورت پوسته نشان می دهد. ما لیست لایه‌ای گره‌ها را در آرگومان **nlist** ذکر می‌کنیم. خروجی نمایش داده شده در شکل ۷-۷ نسبت به خروجی قبلی هم شکل است.



شکل ۷-۷. نمودار پترسن به صورت یک پوسته تصویرسازی شده است

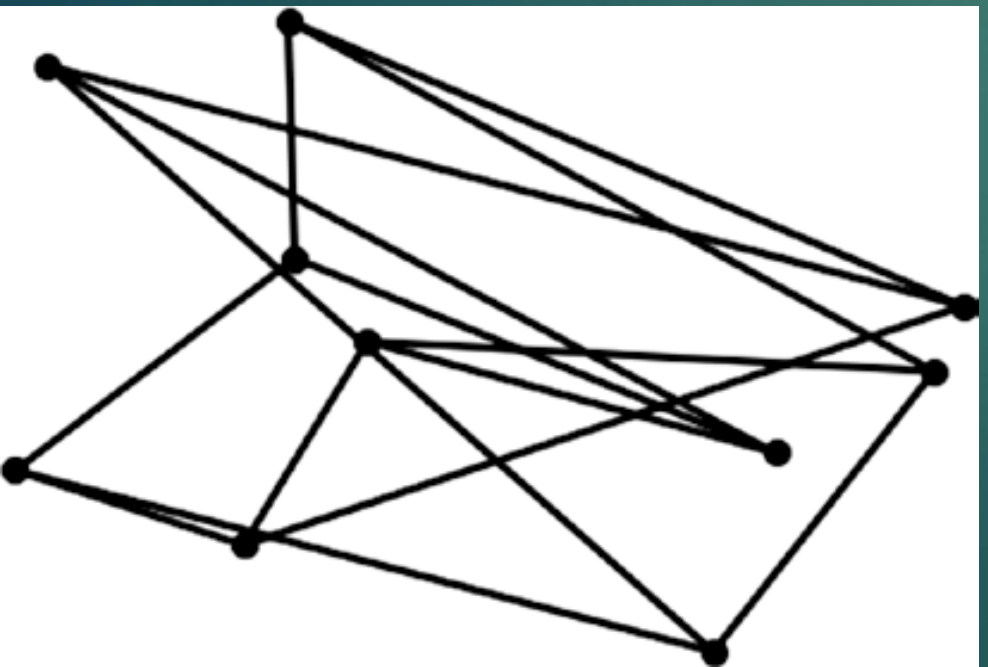
ما می توانیم نحوه نمایش نمودارهایمان را سفارشی کنیم. برای سفارشی کردن اندازه گره ها و عرض خطوط از کد زیر استفاده کنید:

```
options = {'node_color': 'black', 'node_size': 100, 'width': 3}
```

حالا بیایید نمودار پترسن را با این گزینه ها تصویرسازی کنیم:

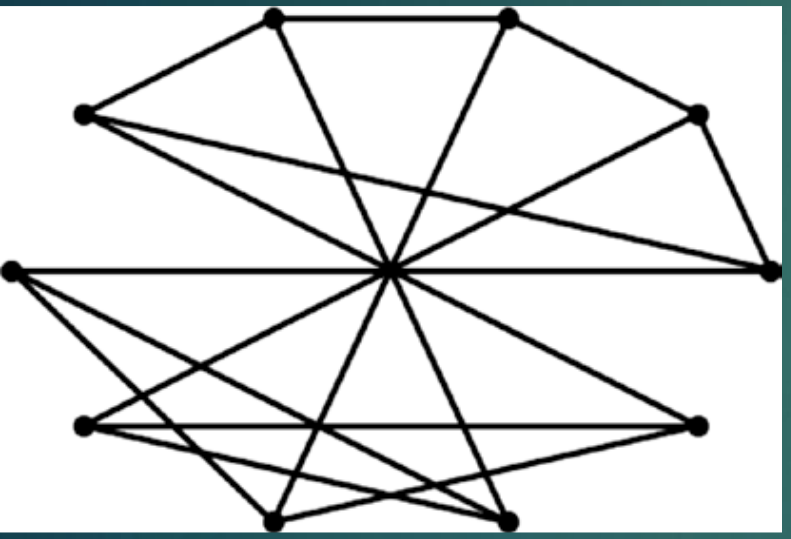
```
nx.draw_random(G, **options)
plt.show()
```

شکل ۷-۸. نمودار پترسن با گره ها و یال های سفارشی شده



در مرحله بعد، می توانیم آن را به صورت دایره ای به صورت زیر تصویرسازی کنیم:

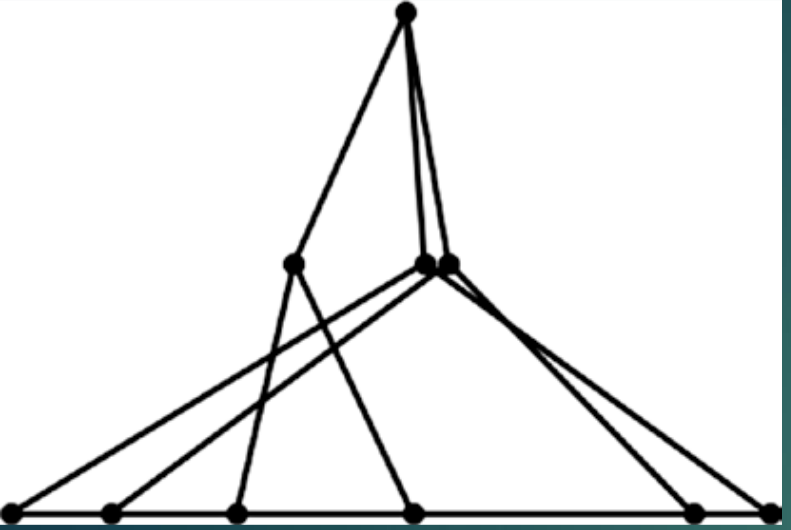
```
nx.draw_circular(G, **options)
plt.show()
```



شکل ۷-۹. نمودار پترسن در پیکربندی دایره ای

همچنین می توانیم نمودار پترسن را در یک پیکربندی طیفی به صورت زیر به تصویر بکشیم:

```
nx.draw_spectral(G, **options)
plt.show()
```

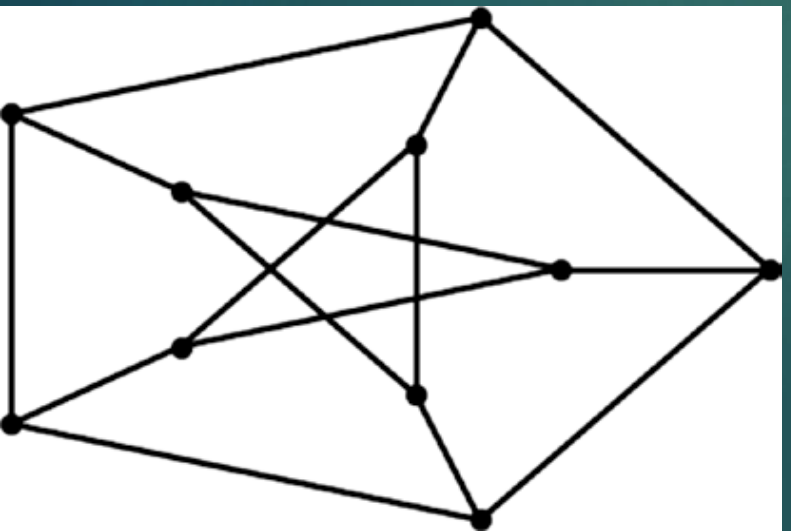


شکل ۷-۱۰. نمودار پترسن در پیکربندی طیفی

می‌توانیم نمودار پترسن را در پیکربندی پوسته با گزینه‌های سفارشی‌شده به صورت زیر به تصویر بکشیم:

```
nx.draw_shell(G, nlist=[range(5, 10), range(5)], **options)
plt.show()
```

شکل ۷-۱۱. گراف پترسن در پیکربندی پوسته با گره ها و یال های سفارشی



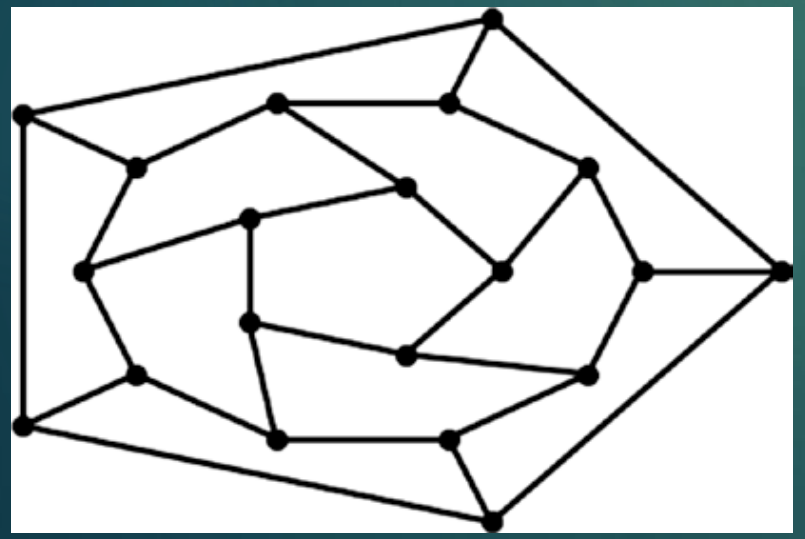
همچنین می توانیم یک نمودار دوازده وجهی به صورت زیر ایجاد کنیم:

```
G = nx.dodecahedral_graph()
```

می توان آن را در قالب پوسته به صورت زیر تصویرسازی کرد:

```
shells = [[2, 3, 4, 5, 6],[8, 1, 0, 19, 18, 17, 16, 15, 14, 7],[9, 10, 11, 12, 13]]
nx.draw_shell(G, nlist=shells, **options)
plt.show()
```

شکل ۷-۱۲. نمودار دوازده وجهی در پیکربندی پوسته با گره ها و یال های سفارشی



با استفاده از این دستور می توانید هر شکل گراف را به عنوان فایل روی دیسک ذخیره کنید:

```
plt.savefig('graph.png’)
```

این عبارت یک فایل را روی دیسک در محل برنامه ذخیره می کند.

اختصاص برچسب های سفارشی به گره ها

می توانید از کد زیر برای اختصاص برچسب های سفارشی به گره ها استفاده کنید:

```
G = nx.path_graph(4)
cities = {0: 'Mumbai', 1: 'Hyderabad’,
          2: 'Banglore', 3: 'New York'}
H=nx.relabel_nodes(G, cities)
print('Nodes of the graph H: ')
print(H.nodes())
print('Edges of the graph H: ')
print(H.edges())
```

خروجی بصورت زیر است:

Nodes of the graph H:

['Mumbai', 'Hyderabad', 'Banglore', 'New York']

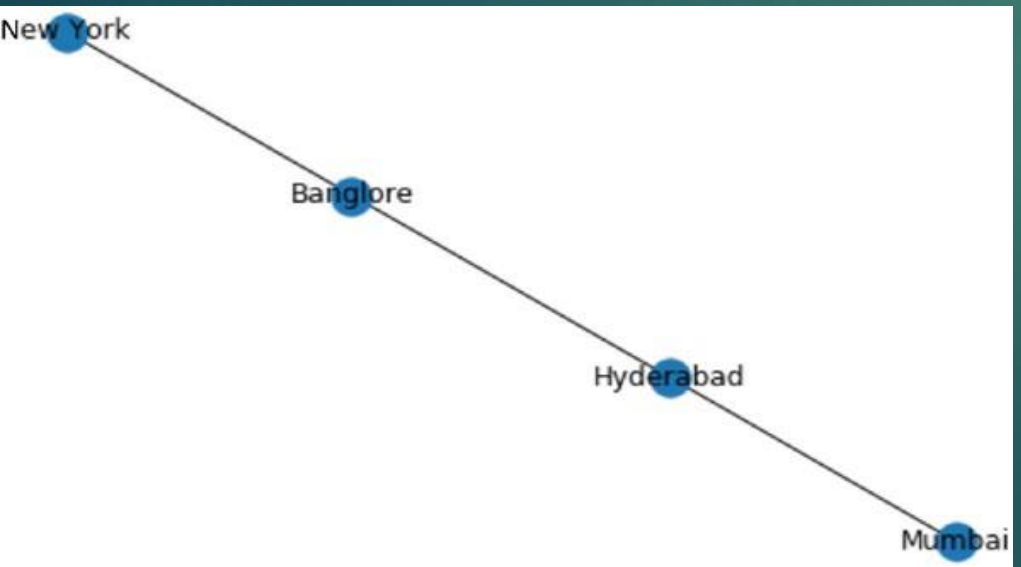
Edges of the graph H:

[('Mumbai', 'Hyderabad'), ('Hyderabad', 'Banglore'),

('Banglore', 'New York')]

از این کد برای ایجاد تصویرسازی استفاده کنید:

```
nx.draw(H, with_labels=True)
plt.show()
```



شکل ۷-۱۳. گره‌های علامت‌گذاری‌شده در یک گراف خطی یا مسیر

خلاصه

این فصل مفهوم نمودارها را بررسی کرده و انواع مختلف را معرفی می کند. شما یاد گرفتید که آنها را به روش های مختلف تصویرسازی کنید. اکنون باید با تصویرسازی ساختارهای داده گراف در پایتون راحت باشید.

فصل بعدی کتابخانه علوم داده SciPy، پانداس را پوشش می دهد. ما ساختارهای داده همه کاره را در کتابخانه، سری و چارچوب داده پانداها به همراه چند نمونه از تصویرسازی داده ها معرفی می کنیم.