دانشکده مهندسی برق

تمرین اول درس هوش مصنوعی

استاد درس:

جناب آقای دکتر علیاری

پریا ساعی ۴۰۱۱۹۱۶۳

ارشیا کلانتریان ۴۰۱۲۱۹۹۳

مریم سلطانی ۴۰۱۱۹۴۳۳

نیمسال دوم ۱۴۰۳_۱۴۰۴

لینک گوگل کولب

https://colab.research.google.com/drive/1Ee0LwUZsVaHWST8YXpUWRDxqV9LoSZ4k?usp=sharing

لینک گیت‌هاب

مریم سلطانی: https://github.com/MaryamSoltani28/AI_2025

ارشیا کلانتریان: https://github.com/ARKAL-J04/MachineLearning2025

پریا ساعی: https://github.com/Paria-s/AI_4032

Subject: Date:

(I) الف) ماتریس ها را در صورتی می‌توان در هم ضرب کرد که تعداد ستون‌های ماتریس اول با سطرهای ماتریس دوم برابر باشد در این صورت ماتریس حاصل ابعادی به شکل (تعداد ستون ماتریس دوم × تعداد سطر ماتریس اول) دارد:

$[A]=(2\times3)$ $[B]=(4\times2)$

$\Rightarrow [BA]=(4\times3)$ $[B^T]=(2\times4)$ $[B^TA]$ $[A^TB]$

ب)
$$BA=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}_{4\times2}\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2\times3}=\begin{bmatrix} b_{11}a_{11}+b_{12}a_{21} \\ b_{21}a_{11}+b_{22}a_{21} \\ b_{31}a_{11}+b_{32}a_{21} \\ b_{41}a_{11}+b_{42}a_{21} \end{bmatrix}$$

(I) الف) ماتریس‌ها را در صورتی می‌توان در هم ضرب کرد که تعداد ستون‌های ماتریس اول با سطرهای ماتریس دوم برابر باشد در این صورت ماتریس حاصل ابعادی به شکل (تعداد ستون ماتریس دوم × تعداد سطر ماتریس اول) دارد.

اگر ابعاد ماتریس A برابر $m\times n$ باشد ابعاد A^T برابر $n\times m$ می‌باشد.

$[A]=2\times3$, $[B]=4\times2$, $[BA]=4\times3$, $[B]^T=2\times4$, $[B^TA]=X$, $[A^TB]=X$

ب)
$$BA=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}_{4\times2}\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2\times3}=\begin{bmatrix} b_{11}a_{11}+b_{12}a_{21} & b_{11}a_{12}+b_{12}a_{22} & b_{11}a_{13}+b_{12}a_{23} \\ b_{21}a_{11}+b_{22}a_{21} & b_{21}a_{12}+b_{22}a_{22} & b_{21}a_{13}+b_{22}a_{23} \\ b_{31}a_{11}+b_{32}a_{21} & b_{31}a_{12}+b_{32}a_{22} & b_{31}a_{13}+b_{32}a_{23} \\ b_{41}a_{11}+b_{42}a_{21} & b_{41}a_{12}+b_{42}a_{22} & b_{41}a_{13}+b_{42}a_{23} \end{bmatrix}_{4\times3}$$

$$B^T=\begin{bmatrix} b_{11} & b_{21} & b_{31} & b_{41} \\ b_{12} & b_{22} & b_{32} & b_{42} \end{bmatrix}_{2\times4}$$

(II) الف)
$$\vec{x}^{(i)}\vec{\theta}=\begin{bmatrix} x_1^{(i)} & x_2^{(i)} \end{bmatrix}_{1\times2}\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}_{2\times1}=\begin{bmatrix} \theta_1x_1^{(i)}+\theta_2x_2^{(i)} \end{bmatrix}_{1\times1}=\theta_1x_1^{(i)}+\theta_2x_2^{(i)}$$

$$X\vec{\theta}=\begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} \end{bmatrix}_{n\times2}\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}_{2\times1}=\begin{bmatrix} x_1^{(1)}\theta_1+x_2^{(1)}\theta_2 \\ x_1^{(2)}\theta_1+x_2^{(2)}\theta_2 \\ \vdots \\ x_1^{(n)}\theta_1+x_2^{(n)}\theta_2 \end{bmatrix}_{n\times1}$$

ب)
$$z^{(i)}=\begin{bmatrix} x_1^{(i)} & x_2^{(i)} \end{bmatrix}\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}-y^{(i)} \quad , \quad \vec{z}=\begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(n)} \end{bmatrix}_{n\times1} \Rightarrow J=\vec{z}^T\vec{z}$$

DAT

$$\nabla_{\theta} J = \frac{\partial J}{\partial \vec{\theta}} = \frac{\partial \sum_{i=1}^{n} (x^{(i)}\vec{\theta} - y^{(i)})^2}{\partial \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}} = \sum_{i=1}^{n} \left(\frac{\partial (\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} - y^{(i)})^2}{\partial \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}} \right)$$

$$= \sum_{i=1}^{n} \begin{pmatrix} 2x_1^{(i)}(\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} - y^{(i)}) \\ 2x_2^{(i)}(\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} - y^{(i)}) \end{pmatrix} = \sum_{i=1}^{n} 2(\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} - y^{(i)}) \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$$

$$= \sum_{i=1}^{n} 2(\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} - y^{(i)}) (x^{(i)})^T$$

ا)

ابتدا فایل داده‌ها در گوگل درایو آپلود شده و در حالت عمومی قرار گرفته سپس با کمک قسمت مورد نظر لینک مربوط به فایل (که در اینجا آورده شده) ، دیتا ها را با دستور مناسب در گوگل کولب بارگذاری می‌کنیم:

```
#https://drive.google.com/file/d/1iMjo4Qa_JQUUcxmcdUV0dWRQWevb_0br/view?usp=sharing
!pip install --upgrade --no-cache-dir gdown
!gdown 1iMjo4Qa_JQUUcxmcdUV0dWRQWevb_0br
```

آ)۱

فرمت فایل: .mat

دستور مورد نیاز برای خواندن فایل: scipy.io.loadmat

آ)۲

داده خوانده شده از کلاس دیکشنری است که شامل تعدادی کلید و مقادیر مربوط به آنها می‌شود.

اجزا:'header', 'version', 'globals', 'X201_DE_time', 'X201_FE_time', 'X201RPM'

```
# خواندن فایل .mat
mat_data = scipy.io.loadmat("/content/201.mat")
# نمایش نوع داده ها
print(type(mat_data))
# نمایش کلیدهای موجود برای یافتن نام متغیرها
print(mat_data.keys())
```

آ)۳

از میان کلید های درون دیکشنر، کلیدی را که مربوط به زمان می‌شود انتخاب می‌کنیم و در متغیری دیگر ذخیره می‌کنیم سپس با کمک دستور پرینت، نوع داده و ابعاد آن را نمایش می‌دهیم.

```
# فرض کنیم داده‌های اصلی در 'X109_DE_time' ذخیره شده باشند
data_array = mat_data["X201_DE_time"]  # ذخیره در متغیر

print(type(data_array))  # بررسی نوع داده
print(data_array.shape)  # نمایش ابعاد داده ها
```

ب)۱

نرخ نمونه برداری که در صورت سوال داده شده را به عنوان فرکانس تعریف کرده

دوره تناوب را از تقسیم یک بر آن محاسبه می‌کنیم

با تابعی که در پایتون برای اندازه گیری طول تعریف شده، طول داده‌ها را محاسبه کرده و آن را در متغیر مربوط به تعداد نمونه‌ها ذخیره می‌کنیم.

در آخر با کمک کتابخانه numpy و دادن پارامتر های شروع ، پایان و تعداد بازه مورد نظر، محور افقی ایجاد می‌شود.

np.linspace : در یک بازه مشخص، اعداد را با فاصله مساوی برمی‌گرداند.

```
Fs = 48000  # نرخ نمونه برداری
T = 1 / Fs  # دوره نمونه‌برداری
N = len(data_array)  # تعداد نمونه ها
time = np.linspace(0, N*T, N)  # ایجاد بردار زمان
```

شکل کلی نمودار ایجاد شده و طول و عرض آن مشخص می‌شود.

داده های مربوط به هر محور تعیین می‌شود. محور افقی زمان و محور عمودی داده‌هایی است که در ابتدا استخراج شده بود و پارامتر آخر نام سیگنال را مشخص می‌کند.

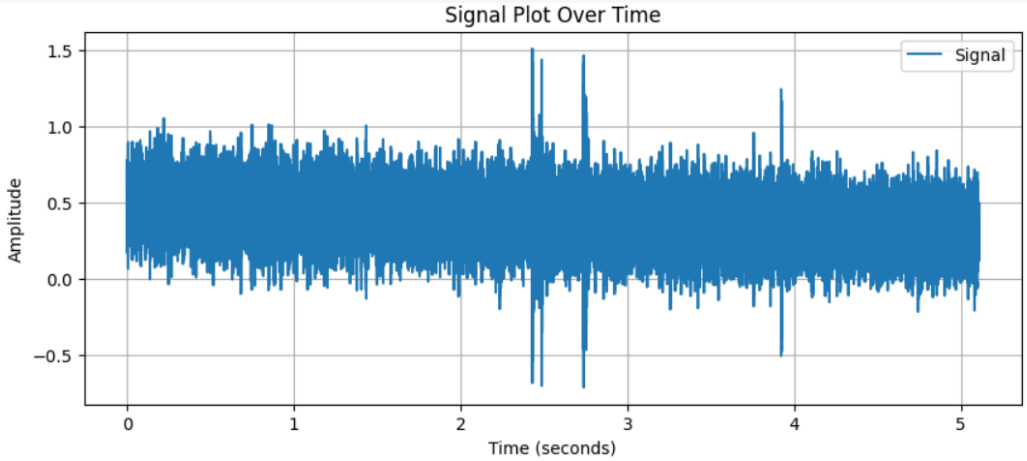
لیبل مربوط به محور ها و عنوان نمودار تعیین می‌شود.

plt.legend() : راهنمای نمودار را تشکیل می‌دهد که در اینجا تنها یک سیگنال داریم که لیبل آن در خط های قبلی مشخص شده بود

plt.grid(True) : صفحه نمودار را تقسیم بندی می‌کند.

plt.show(): سیگنال را نمایش می‌دهد.

```
plt.figure(figsize=(10, 4))
plt.plot(time, data_array, label="Signal")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.title("Signal Plot Over Time")
plt.legend()
plt.grid(True)
plt.show()
```



مشاهده می‌شود در اکثر زمان‌ها سیگنال مورد نظر دامنه‌ای بین ۰ و ۱ دارد. تعداد نوسانات در واحد زمان خیلی زیاد بوده که نشان دهنده فرکانس بالای سیگنال است.

ب)۲

یک فیلتر تعریف می‌کنیم که زمان های بزرگتر از ۲ ثانیه و کوچکتر از ۲,۰۱ ثانیه را برای ما جدا کند. در واقع یک عبارت بر اساس جبر بولی نوشته‌ایم که تنها زمانی درست است که هردو جمله مقدار یک منطقی داشته باشند.

آرایه زمان را با فیلتری که ایجاد کردیم، اندیس گذاری می‌کنیم با این کار تنها مقادیری از زمان انتخاب می‌شوند که در آن عبارت جبر بولی مربوط به فیلتر مقدار یک داشته باشد و در نتیجه بازه زمانی مورد نظر ایجاد می‌شود.

همانند کاری که برای زمان انجام دادیم را برای آرایه داده‌ها انجام می‌دهیم و در واقع مقادیر سیگنال را فیلتر می‌کنیم.

```
# انتخاب نقاطی که در بازه ۲ تا ۲,۰۱ ثانیه هستند
mask = (time >= 2) & (time <= 2.01)  # ایجاد ماسک فیلتر
time_filtered = time[mask]  # انتخاب مقادیر مربوط به این بازه
signal_filtered = data_array[mask]  # انتخاب داده های سیگنال در این بازه
```

شکل کلی نمودار ایجاد شده و طول و عرض آن مشخص می‌شود.

داده های مربوط به هر محور تعیین می‌شود. محور افقی زمان فیلتر شده و محور عمودی داده‌های فیلتر شده و دو پارامتر آخر نام سیگنال و رنگ آن را مشخص می‌کند.

لیبل مربوط به محور ها و عنوان نمودار تعیین می‌شود

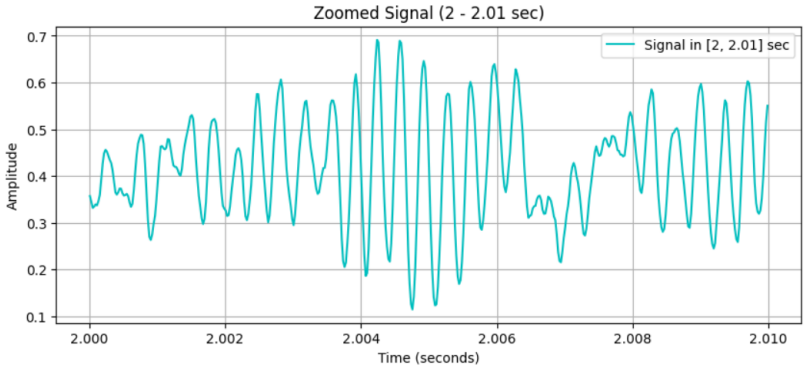
plt.legend() : راهنمای نمودار را تشکیل می‌دهد که در اینجا تنها یک سیگنال داریم که لیبل آن در خط های قبلی مشخص شده بود

plt.grid(True) : صفحه نمودار را تقسیم بندی می‌کند.

plt.show() : سیگنال را نمایش می‌دهد.

```
plt.figure(figsize=(10, 4))
plt.plot(time_filtered, signal_filtered, label="Signal in [2, 2.01] sec", color="c")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.title("Zoomed Signal (2 - 2.01 sec)")
plt.legend()
plt.grid(True)
plt.show()
```

در این بخش سیگنال را به صورت زوم شده در بازه زمانی ۲ تا ۲,۰۱ ثانیه مشاهده می‌کنیم.



ج)۱و۲

یک تابع تعریف می‌کنیم که در ورودی سیگنال حوزه زمان و نرخ نمونه برداری را گرفته و در خروجی، طیف فرکانسی آن را نمایش داده و فرکانس غالب را تعین می‌کند.

همانطور که قبل تر محاسبه شد، تعداد نمونه ها را بدست می‌آوریم.

تبدیل فوریه گسسته سیگنال را به کمک تابع از پیش تعریف شده در کتابخانه numpy محاسبه کرده و ضرایب تبدیل فوریه در یک متغیر ذخیره می‌شود.

fft_magnitude = np.abs(fft_values) / N : مقدار مطلق ضرایب تبدیل فوریه را محاسبه می‌کند و دامنه را نرمال‌سازی می‌کند تا مستقل از تعداد نمونه‌ها باشد.

frequencies = np.fft.fftfreq(N, 1 / Fs) : فرکانس‌های مربوط به هر ضریب را محاسبه می‌کند و سپس با داشتن زمان بین دو نمونه متوالی،محور فرکانسی را برای طیف فرکانسی محاسبه می‌کند.

half_N = N // 2: تعداد نمونه‌ها را بر دو تقسیم کرده و خروجی ای به صورت عدد صحیح دارد.

frequencies = frequencies[:half_N] : فقط بخش مثبت فرکانس‌ها را در نظر می‌گیرد (زیرا طیف فرکانسی متقارن است).

fft_magnitude = fft_magnitude[:half_N] : این خط دامنه مربوط به بخش مثبت فرکانس‌ها را انتخاب می‌کند.

dominant_index = np.argmax(fft_magnitude) : بزرگترین مقدار در دامنه تبدیل فوریه سیگنال را پیدا کرده و اندیس مربوط به آن را برمی‌گرداند.

dominant_frequency = frequencies[dominant_index] : با داشتن اندیس بزرگترین دامنه، فرکانس مربوط به آن را از میان فرکانس‌ها بدست می‌آورد.

شکل کلی نمودار ایجاد شده و طول و عرض آن مشخص می‌شود.

داده های مربوط به هر محور تعیین می‌شود. محور افقی فرکانس‌ها و محور عمودی تبدیل فوریه سیگنال و دو پارامتر آخر نام سیگنال و رنگ آن را مشخص می‌کند.

plt.axvline : این تابع از کتابخانه matplotlib.pyplot برای رسم یک خط عمودی استفاده می‌شود.

axvline : مخفف "axis vertical line"

x=dominant_frequency : این پارامتر موقعیت خط عمودی را روی محور فرکانس تعیین می‌کند.

رنگ و استایل و لیبل خط تعیین می‌شود و فرکانس غالب مشخص می‌شود.

لیبل مربوط به محور ها و عنوان نمودار تعیین می‌شود.

plt.legend() : راهنمای نمودار را تشکیل می‌دهد که در اینجا تنها یک سیگنال داریم که لیبل آن در خط های قبلی مشخص شده بود.

plt.grid(True) : صفحه نمودار را تقسیم بندی می‌کند.

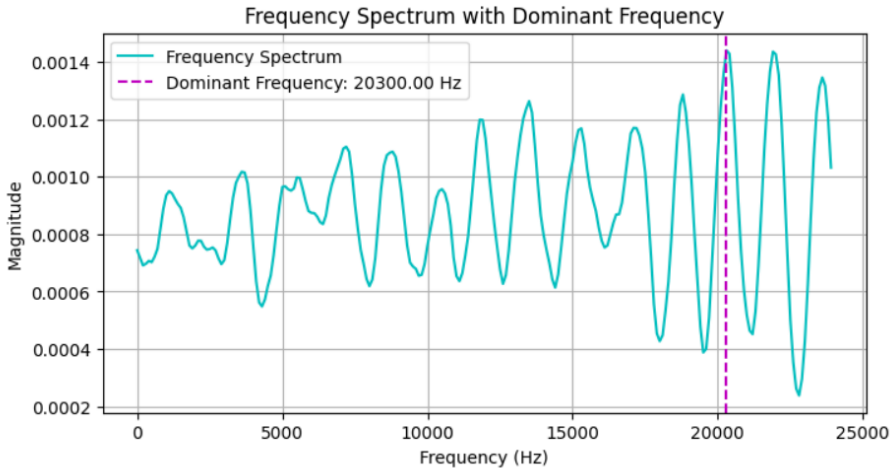
plt.show() : سیگنال را نمایش می‌دهد.

فرکانس غالب به صورت عدد اعشاری با دو رقم اعشار در خروجی چاپ می‌شود.

```
def plot_frequency_spectrum_with_dominant(signal, Fs):

    N = len(signal)  # تعداد نمونه‌ها
    fft_values = np.fft.fft(signal)  # محاسبه FFT
    fft_magnitude = np.abs(fft_values) / N  # نرمال‌سازی دامنه
    frequencies = np.fft.fftfreq(N, 1 / Fs)  # محاسبه محور فرکانسی

    # فقط بخش مثبت فرکانس‌ها را در نظر بگیریم
    half_N = N // 2
    frequencies = frequencies[:half_N]
    fft_magnitude = fft_magnitude[:half_N]

    # پیدا کردن اندیس فرکانس غالب
    dominant_index = np.argmax(fft_magnitude)
    dominant_frequency = frequencies[dominant_index]

    # رسم طیف فرکانسی
    plt.figure(figsize=(8, 4))
    plt.plot(frequencies, fft_magnitude, color='c', label="Frequency Spectrum")
    plt.axvline(x=dominant_frequency, color='m', linestyle='--', label=f"Dominant
Frequency: {dominant_frequency:.2f} Hz")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.title("Frequency Spectrum with Dominant Frequency")
    plt.legend()
    plt.grid(True)
    plt.show()

    print(f"🔷 dominant: {dominant_frequency:.2f} Hz")
```

سیگنال فیلتر شده و نرخ نمونه برداری را به عنوان ورودی به تابع قسمت قبل می‌دهیم:

```
plot_frequency_spectrum_with_dominant(signal_filtered, Fs)
```



◆ dominant: 20300.00 Hz

باتوجه به نمودار خروجی می‌توان دید که فرکانس غالب که مربوط به بیشترین دامنه است در حدود ۲ کیلوهرتز بوده و مقدار دقیق آن زیر نمودار ذکر شده است.

د)

در این بخش از ما خواسته شده است که سیگنال را به قطعه هایی با اندازه ۱۲۸ تقسیم کرده و این قطعه ها هم پوشانی داشته باشند و در نهایت هر قطعه را در یک آرایه numpy ذخیره کنیم

برای انجام این کار نیازمند تعریف یک تابع هستیم که سیگنال را به عنوان ورودی دریافت کرده و با تقسیم سیگنال به قطعه هایی به اندازه ۱۲۸، هر قسمت را به عنوان نمونه در لیستی ذخیره کرده و با درنظر گرفتن اندازه همپوشانی یک گام جلو برود و ۱۲۸ نمونه بعدی را در لیست ذخیره کند

از آنجایی که ما در این پردازش اندازه همپوشانی ۳۲ یا ۲۵٪ است پس در نتیجه اندازه گام ما نیز باید $128 - 32 = 96$ باشد.

step = segment_size - overlap

سپس لیستی را برای ذخیره هر قطعه ایجاد میکنیم

segments = []

حال میخواهیم به درستی این قطعه ها را جدا کرده و سپس وارد لیست ایجاد شده کنیم برای اینکار از حلقه for کمک میگیریم که از نقطه اول سیگنال که ۰ است شروع کرده و به اندازه گام (۱۲۸) جلو رفته و تا آخرین مکانی که میتواند نمونه برداری کند پیش میرود

برای اینکه نمونه آخر حفظ شود ما باید یک را به اندازه سیگنال اضافه کنیم

len(signal) - segment_size + 1

سپس تعیین میکنیم نقطه انتهای هر قطعه چطور بدست می آید که برابر است با نقطه شروع هر قطعه به علاوه اندازه گام

end_index = start_index + segment_size

حال این قطعه را از سیگنال اصلی جدا میکنیم

segment = signal[start_index:end_index]

و بعد به لیست ایجاد شده اضافه میکنیم

segments.append(segment)

حال میخواهیم این لیست را به آرایه numpy تبدیل کنیم اینکار به منظور پردازش سریعتر و راحت تر میکند خصوصا برای ماتریس های چند بعدی، و بعد این مقدار را باید برگرداند و تحویل دهد

return np.array(segments)

پس تابع را به صورت زیر تعریف می‌شود

```
def split_signal_with_overlap(signal, segment_size=128, overlap=32): #25% همپوشانی

    step = segment_size - overlap  # اندازه گام (با توجه به همپوشانی)
    segments = []  # لیستی برای ذخیره قطعه‌ها

    # تقسیم سیگنال به قطعه‌ها با همپوشانی
    for start_index in range(0, len(signal) - segment_size + 1, step):
        end_index = start_index + segment_size
        segment = signal[start_index:end_index]
        segments.append(segment)  # اضافه کردن قطعه به لیست

    return np.array(segments)  # تبدیل لیست به آرایه NumPy
```

و در نهایت با تابع تعریف شده در قسمت بالا، سیگنال اصلی را به این تابع داده و خروجی را بدست می‌آوریم

```
segments = split_signal_with_overlap(data_array, segment_size=128, overlap=32)

print(f"تعداد قطعه‌ها: {segments.shape[0]}")
print(f"ابعاد هر قطعه: {segments.shape[1]}")
print(segments)  # نمایش قطعه‌ها
```

ه)

۱- در این بخش ابتدا میخواهیم نمونه های حاصله در بخش د را در یک pandas.DataFrame ذخیره کنیم

اینکار به منظور ذخیره داده ها است و به ما قابلیت دیدن شکل سیگنال های جدا شده را می دهد

پس ابتدا ما باید ابتدا سیگنال را دو بعدی سازی کنیم که بتوانیم نمودار آن در pandas.DataFrame ذخیره کنیم چون در غیر این صورت پذیرفته نمیشود

```
segments_2d = np.squeeze(segments)
```

سپس داده ها را در دیتافریم ذخیره میکنیم

```
df = pd.DataFrame(segments_2d)
```

در نتیجه کد به صورت زیر خواهد بود:

```
segments_2d = np.squeeze(segments)
df = pd.DataFrame(segments_2d)  # تبدیل آرایه به DataFrame
df
```

۲- حالا میخواهیم به کمک دیتافریم ایجاد شده ۱۰ قطعه سیگنال مضرب ۱۳ را روی یک شکل نشان دهیم

برای اینکار ابتدا باید داده های دو بعدی را به صورت یک جدول درآورده

```
df = pd.DataFrame(segments_2d)
```

می دانیم df به صورت ماتریسی با ۱۲۸ ستون که طول هر نمونه بوده و ۲۵۵۳ سطر که بیانگر هر نمونه بوده، حال میخواهیم ۱۰ تا سیگنال نمونه مضرب ۱۳ را انتخاب کنیم

```
num_segments = 10
```

```
indices = [i * 13 for i in range(num_segments)]
```

سپس هدف ما، ایجاد رنگ ها مختلف است که هر سیگنال رنگ مخصوص به خود را داشته باشد، برای رنگ ها از دستور tab10 استفاده میکنیم که یک پالت ۱۰ رنگی آماده است و هر رتگ را به یک سیگنال اختصاص میدهد

```
colors = plt.cm.get_cmap("tab10", num_segments)
```

حال باید سیگنال نمایش دهیم که برای اینکار ابتدا اندازه نمودار را مشخص کرده

```
plt.figure(figsize=(10, 5))
```

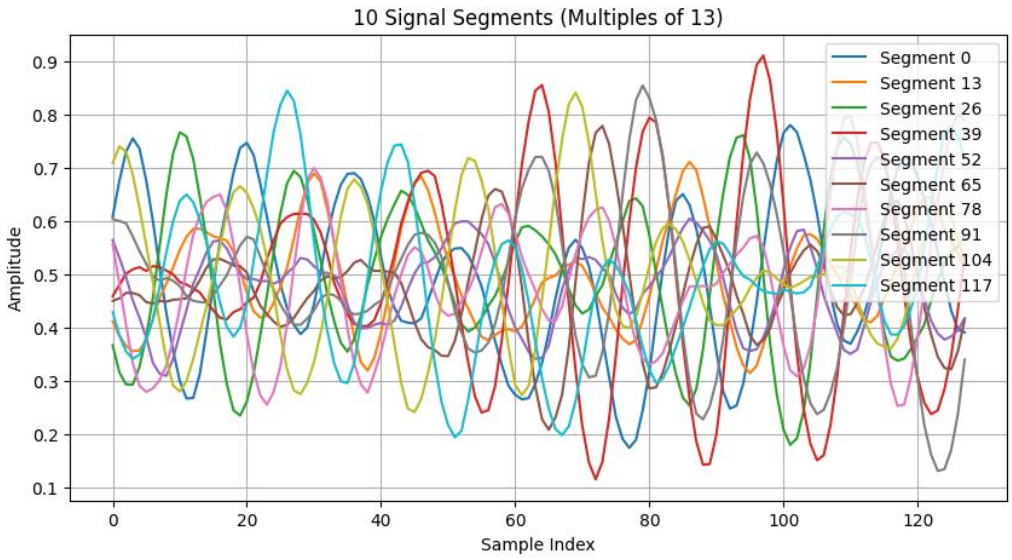
حال برای نشان دادن هر سیگنال از یک حلقه for استفاده کرده و هر ۱۰ مقدار ضریب ۱۳ از قبل ذخیره شده در indices را دونه دونه نمایش میدهیم که در آن مقدار idx بیانگر هر سطر مضرب ۱۳ است که به کمک df.iloc مقادیر آن را از دیتافریم گرفته که همان ۱۰ قطعه انتخابی ما هستند و در نهایت برای هر یک از سیگنال های انتخابی یک Label درنظر می گیریم

plt.plot(df.columns, df.iloc[idx], label=f"Segment {idx}"

در جدول نمایشی نهایی محور افقی باید نمایانگر هر سیگنال باشد و محور عمودی دامنه آن سیگنال را نشان دهد و لیبل هر نمونه باید مشخص باشد بنابراین کد نهایی ما به شکل زیر خواهد بود:

```
df = pd.DataFrame(segments_2d)  # تبدیل آرایه به DataFrame

# انتخاب ۱۰ ردیف که شماره‌ی آن‌ها مضرب ۱۳ است
num_segments = 10
indices = [i * 13 for i in range(num_segments)]  # 10 اول ۱۳ مضرب: [0, 13, 26, ...,
117]

# رنگ‌های مختلف برای نمایش نمودارها
colors = plt.cm.get_cmap("tab10", num_segments)  # استفاده از رنگ‌های آماده

# رسم نمودار
plt.figure(figsize=(10, 5))  # تنظیم اندازه‌ی نمودار
for i, idx in enumerate(indices):
    plt.plot(df.columns, df.iloc[idx], label=f"Segment {idx}", color=colors(i)) #
iloc[i]= سطر iام

plt.xlabel("Sample Index")
plt.ylabel("Amplitude")
plt.title("10 Signal Segments (Multiples of 13)")
plt.legend()  # نمایش لیبل‌ها
plt.grid(True)
plt.show()
```

و شکل نمودار خروجی آن یه صورت زیر است:



II

آ) ۱)

مجموعه داده Iris یکی از مشهورترین مجموعه داده‌ها در یادگیری ماشین و آمار است که برای طبقه‌بندی و تحلیل داده‌ها مورد استفاده قرار می‌گیرد. این مجموعه داده توسط **رونالد فیشر** در سال ۱۹۳۶ معرفی شد و شامل اطلاعات مربوط به سه گونه مختلف گل زنبق (Iris) است.

ویژگی‌های مجموعه دادهIris

مجموعه داده Iris شامل **۱۵۰ نمونه** از سه گونه مختلف گل زنبق است:

۱. **Iris Setosa**

۲. **Iris Versicolor**

۳. **Iris Virginica**

ویژگی‌های ورودی(Feature)

هر نمونه دارای **۴ ویژگی عددی مشخصه یاFeature** است که ویژگی‌های فیزیکی گل را اندازه‌گیری می‌کنند:

طول کاسبرگ (Sepal Length) برحسب سانتی‌متر

عرض کاسبرگ (Sepal Width) برحسب سانتی‌متر

طول گلبرگ (Petal Length) برحسب سانتی‌متر

عرض گلبرگ (Petal Width) برحسب سانتی‌متر

برچسب خروجی(Label)

هر نمونه به یکی از سه کلاس (گونه‌های گل) تعلق دارد:

- **0 → Iris Setosa**
- **1 → Iris Versicolor**
- **2 → Iris Virginica**

چرا مجموعه داده Iris محبوب است؟

۱. **سادگی و توازن داده‌ها :**این مجموعه داده کاملاً متوازن است (هر کلاس ۵۰ نمونه دارد).

۲. **اندازه کوچک :**فقط ۱۵۰ نمونه دارد و برای آزمایش سریع الگوریتم‌های یادگیری ماشین مناسب است.

۳. **قابل فهم بودن :**دارای ویژگی‌های عددی مشخص و بدون داده‌های ناموجود (Missing Data) است.

تحلیل آماری ساده روی مجموعه داده‌Iris

- معمولاً از **نمودار جعبه‌ای (Box Plot)** یا **هیستوگرام (Histogram)** برای بررسی توزیع داده‌ها استفاده می‌شود.
- می‌توان از **نمودار پراکندگی (Scatter Plot)** برای مشاهده جدایی کلاس‌ها بر اساس دو ویژگی خاص بهره برد.

کاربردهای مجموعه داده‌Iris

- **آزمایش الگوریتم‌های طبقه‌بندی** مانند-k نزدیک‌ترین همسایه(KNN) ، درخت تصمیم(Decision Tree) ، و شبکه‌های عصبی مصنوعی.(ANN)
- **یادگیری روش‌های پیش‌پردازش داده‌ها** مانند استانداردسازی و نرمال‌سازی.
- **آزمون الگوریتم‌های کاهش ابعاد** مانند.PCA

آ)۲)

با استفاده از کتابخانه sklearn این دیتاست رو فراخوانی می‌کنیم:

```
from sklearn import datasets
iris = load_iris()
```

آ)۳)

در این قسمت قصد داریم داده‌ها را به دو بخش test و train تقسیم کنیم. برای این کار ابتدا تابع train_test_splitرا از ماژول sklearn.model_selectionفراخوانی می‌کند.

این تابع برای **تقسیم مجموعه داده** به دو بخش **آموزش** و **آزمون** استفاده می‌شود.

```
from sklearn.model_selection import train_test_split
# ۲۰٪) تقسیم داده‌ها به آموزش (۸۰٪) و آزمون
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.2, random_state=63
)
```

کد X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=63)چهار مجموعه مختلف را برمی‌گرداند:

X_trainشامل داده‌های آموزشی که مدل با استفاده از آن یادگیری را انجام می‌دهد.

X_testشامل داده‌های آزمون که برای ارزیابی عملکرد مدل استفاده می‌شود.

y_trainشامل برچسب‌های کلاس مربوط به داده‌های آموزشی است.

y_testشامل برچسب‌های کلاس مربوط به داده‌های آزمون است.

ورودی train_test_splitشامل چندین پارامتر است:

iris.dataویژگی‌های ورودی مجموعه داده را مشخص می‌کند که شامل چهار ویژگی طول و عرض کاسبرگ و گلبرگ است.

iris.targetبرچسب‌های کلاس را مشخص می‌کند که سه مقدار مختلف ۰ و ۱ و ۲ دارد و نشان‌دهنده سه گونه مختلف گل زنبق است.

test_size=0.2تعیین می‌کند که ۲۰ درصد داده‌ها برای آزمون و ۸۰ درصد برای آموزش در نظر گرفته شوند.

random_state=63مقدار تصادفی را ثابت نگه می‌دارد تا داده‌ها در هر اجرا به همان شکل قبلی تقسیم شوند.

پس از اجرای این کد مجموعه داده به دو بخش تقسیم می‌شود که شامل ۱۲۰ نمونه برای آموزش و ۳۰ نمونه برای آزمون خواهد بود.

آ)4

در این بخش قصد داریم داده ها را به یک DataFrame.pandas تبدیل کرده و نام ستون‌ها و نوع دیتا (آموزش یا آزمون) را نیز مشخص کنیم. به شکل زیر عمل می‌کنیم:

ابتدا کتابخانه pandas را Import می‌کنیم.

```
# ۳. دریافت نام ویژگی‌ها
columns = iris.feature_names

# ۴. ایجاد DataFrame برای مجموعه آموزش
df_train = pd.DataFrame(X_train, columns=columns)
df_train["species"] = y_train  # اضافه کردن برچسب کلاس
df_train["dataset_type"] = "train"  # مشخص کردن داده های آموزش

# ۵. ایجاد DataFrame برای مجموعه آزمون
df_test = pd.DataFrame(X_test, columns=columns)
df_test["species"] = y_test  # اضافه کردن برچسب کلاس
df_test["dataset_type"] = "test"  # مشخص کردن داده های آزمون
```

ابتدا نام ویژگی‌های مجموعه داده از iris.feature_namesدریافت می‌شود و در متغیر columnsذخیره می‌شود. سپس برای داده‌های آموزشی یک DataFrame به نام df_trainایجاد می‌شود. در این dataframe ابتدا X_train قرار می‌گیرد که شامل ویژگی‌های ورودی است.

با استفاده از columns=columns نام ستون‌ها به DataFrame اختصاص داده می‌شود.

پس از آن، برچسب‌های کلاس y_train به عنوان یک ستون جدید به نام "species" به df_train اضافه می‌شود. این برچسب‌ها مشخص می‌کنند که هر نمونه به کدام کلاس از گل‌های زنبق تعلق دارد.

ستون "dataset_type" با مقدار "train" اضافه می‌شود تا مشخص شود که این داده‌ها مربوط به مجموعه آموزش هستند.

فرایند مشابهی برای مجموعه آزمون انجام می‌شود. ابتدا X_test در قالب یک DataFrame به نام df_test ذخیره می‌شود. سپس برچسب‌های y_test به عنوان ستون "species" اضافه می‌شوند و در نهایت ستون "dataset_type" با مقدار "test" اضافه می‌شود تا نشان دهد که این داده‌ها متعلق به مجموعه آزمون هستند.

در نهایت df_train و df_test شامل تمام ویژگی‌ها، برچسب کلاس و نوع مجموعه داده (آموزشی یا آزمون) خواهند بود.

پس به صورت کلی:

وقتی X_train را در df_train قرار می‌دهیم، این یعنی **ویژگی‌های ورودی مجموعه داده** را در یک DataFrame ذخیره می‌کنیم. مجموعه داده iris دارای ۴ ویژگی عددی برای هر نمونه است که به ترتیب عبارت‌اند از:

- **sepal length (cm)** طول کاسبرگ
- **sepal width (cm)** عرض کاسبرگ
- **petal length (cm)** طول گلبرگ
- **petal width (cm)** عرض گلبرگ

پس X_train یک ماتریس است که شامل ۸۰٪ نمونه‌های داده همراه با این ۴ ویژگی است

بعد از این که X_train را داخل یک DataFrame ذخیره کردیم، **دو ستون جدید** هم به آن اضافه کردیم

species. شامل y_train که برچسب کلاس (۰، ۱، ۲) را نشان می‌دهد و مشخص می‌کند که هر نمونه مربوط به کدام گونه گل زنبق است. **dataset_type** که مقدار "train" دارد و نشان می‌دهد که این داده‌ها مربوط به مجموعه آموزش هستند.

همین کار برای داده‌های آزمون (X_test) انجام شده است و ستون "dataset_type" در آن مقدار "test" دارد.

آ)۵)

در این قسمت قصد داریم دو DataFrame آموزش و آزمون را با هم ادغام کنیم.

```
df_final = pd.concat([df_train, df_test], ignore_index=True)

print(df_final.head(150))
```

در ابتدا دو **DataFrame** که قبلاً ساخته شده‌اند df_train و df_test با استفاده از pd.concat ادغام می‌شوند و در متغیر df_final ذخیره می‌شوند.

پارامتر ignore_index=True باعث می‌شود که **ایندکس‌های جدید** برای DataFrame نهایی ساخته شود و شماره ایندکس‌های قبلی نادیده گرفته شوند. (دلیل این کار این است که وقتی دو DataFrame df_train و df_test را ترکیب می‌کنیم، هر کدام ایندکس‌های مخصوص به خودشان دارند که از مجموعه اصلی گرفته شده‌اند. اگر این ایندکس‌ها حفظ شوند، ممکن است ایندکس‌های تکراری یا نامرتب در df_finalایجاد شود)

پس از ادغام، df_final شامل تمامی نمونه‌های آموزشی و آزمون خواهد بود که هر نمونه دارای چهار ویژگی اصلی مجموعه داده **Iris**، یک ستون برچسب کلاس (species) و یک ستون نشان‌دهنده نوع مجموعه داده (dataset_type) است.

در نهایت، print(df_final.head(150)) نمایش می‌دهد که **تمام ۱۵۰ نمونه** از مجموعه داده **Iris** در df_final قرار دارند و خروجی را چاپ می‌کند.

در نهایت df_final شامل ۶ ستون خواهد بود:

۱) sepal length (cm) .
۲) sepal width (cm) .
۳) petal length (cm) .
۴) petal width (cm) .
۵) species .برچسب کلاس
۶) dataset_type .مشخص‌کننده نوع داده train یا test

با این کار، کل داده‌های آموزش و آزمون را در یک **DataFrame** یکپارچه ذخیره می‌کنیم که نمایش و پردازش آن راحت‌تر خواهد بود.

ب) ۱)

در این قسمت می‌خواهیم دو ویژگی دلخواه انتخاب کرده و پراکندگی نمونه‌ها را به صورت دوبعدی و با تفکیک نوع گل زنبق نمایش دهیم.

برای قسمت ب ابتدا این کتابخانه را import می‌کنیم:

```
import seaborn as sns
```

سپس:

```
# ۱. بارگیری مجموعه داده Iris
iris = load_iris()

# ۲. تبدیل داده‌ها به DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["species"] = iris.target  # اضافه کردن برچسب کلاس
df["species"] = df["species"].map({0: "Setosa", 1: "Versicolor", 2: "Virginica"})  # نامگذاری کلاس‌ها

# ۳. رسم نمودار پراکندگی (Scatter Plot)
plt.figure(figsize=(8, 6))  # تنظیم اندازه نمودار
sns.scatterplot(
    x=df["sepal length (cm)"],
    y=df["sepal width (cm)"],
    hue=df["species"],  # تفکیک رنگ‌ها بر اساس نوع گل
    palette=["red", "blue", "green"],  # رنگ‌بندی کلاس‌ها
    s=70  # تنظیم اندازه نقاط
)

# ۴. تنظیمات نمودار
plt.xlabel("Sepal Length (cm)")  # برچسب محور x
plt.ylabel("Sepal Width (cm)")  # برچسب محور y
plt.title("Scatter Plot of Sepal Length vs. Sepal Width")  # عنوان نمودار
plt.legend(title="Species")  # نمایش راهنمای رنگ‌ها
plt.grid(True)  # افزودن خطوط شبکه

# ۵. نمایش نمودار
plt.show()
```

ابتدا مجموعه داده با استفاده از load_iris() بارگیری شده و در متغیر iris ذخیره می‌شود.

سپس iris.data که شامل ویژگی‌های عددی است، در قالب یک DataFrame به نام df ایجاد می‌شود. نام ستون‌های این DataFrame همان iris.feature_names است که شامل نام ویژگی‌های مجموعه داده می‌شود.

ستون جدیدی به نام "species" به DataFrame اضافه می‌شود که مقدار آن برابر با iris.target است. این مقدار در ابتدا فقط شامل اعداد ۰، ۱، ۲ است که نشان‌دهنده کلاس‌های مختلف گل‌ها هستند.

برای تبدیل این اعداد به نام‌های واقعی گونه‌ها، از df["species"].map({0: "Setosa", 1: "Versicolor", 2: "Virginica"}) استفاده می‌شود که مقادیر ۰، ۱، ۲ را به "Setosa"، "Versicolor" و "Virginica" تبدیل می‌کند.

در ادامه، نمودار پراکندگی دو ویژگی **طول کاسبرگ** و **عرض کاسبرگ** رسم می‌شود sns.scatterplot() .نقاط داده را روی نمودار نمایش می‌دهد. ویژگی hue=df["species"] باعث می‌شود که نقاط بر اساس گونه‌های مختلف گل، رنگ‌های متفاوتی داشته باشند.

تابع sns.scatterplot() از کتابخانه Seaborn برای رسم نمودار پراکندگی استفاده می‌شود. این نمودار کمک می‌کند تا رابطه بین دو ویژگی عددی را بررسی کنیم. در این کد، هدف نمایش رابطه بین طول کاسبرگ (Sepal Length) و عرض کاسبرگ (Sepal Width)برای سه گونه مختلف گل زنبق است.

پارامترهای استفاده‌شده در sns.scatterplot():

x=df["sepal length (cm)"] .مقدار محور افقی (X) را تعیین می‌کند که طول کاسبرگ است.

y=df["sepal width (cm)"] .مقدار محور عمودی (Y) را تعیین می‌کند که عرض کاسبرگ است

hue=df["species"] .این گزینه باعث می‌شود که نقاط موجود در نمودار بر اساس گونه‌های مختلف گل رنگ‌های متفاوتی داشته باشند.

گونه **Setosa** یک رنگ دریافت می‌کند، گونه **Versicolor** یک رنگ دیگر، گونه **Virginica** یک رنگ سوم.

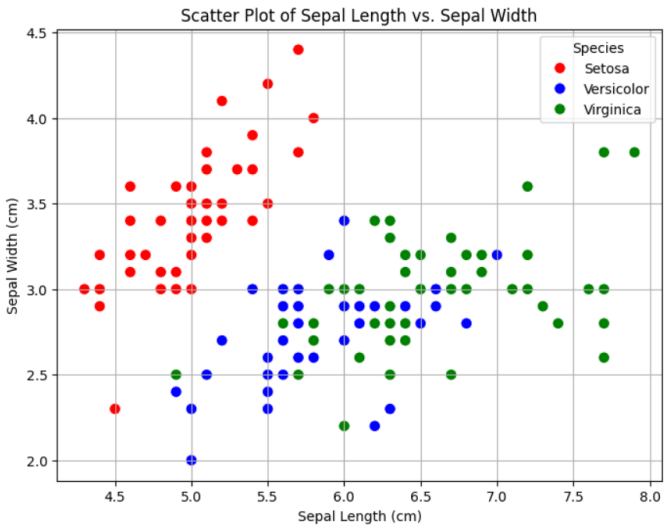
برای رنگ‌بندی، از palette=["red", "blue", "green"] استفاده شده که کلاس‌های مختلف را به ترتیب به رنگ‌های قرمز، آبی و سبز نمایش می‌دهد s=70 .نیز اندازه نقاط داده را تعیین می‌کند.

در بخش تنظیمات نمودار، plt.xlabel() و plt.ylabel() برچسب‌های محورهای x و y را مشخص می‌کنند plt.title() . عنوان نمودار را تعیین می‌کند plt.legend(title="Species") .باعث نمایش راهنمای رنگ‌ها می‌شود plt.grid(True) .نیز خطوط شبکه‌ای را به نمودار اضافه می‌کند.

plt.figure(figsize=(8, 6))یعنی نمودار با اندازه ۸ اینچ عرض و ۶ اینچ ارتفاع ایجاد شود .مقدار ۸ و ۶ فقط نسبت‌های ابعاد نمودار را تعیین می‌کنند و واحد آن اینچ است.

اگر این دستور را حذف کنیم، نمودار با اندازه پیش‌فرض نمایش داده می‌شود که ممکن است خیلی کوچک یا خیلی بزرگ باشد. این تنظیم کمک می‌کند تا نمودار متناسب با صفحه نمایش داده شود.

در نهایت plt.show()باعث نمایش نمودار می‌شود.



ب) ۲)

ابتدا برای نمودار سه بعدی کتابخانه زیر را import می‌کنیم.

```
from mpl_toolkits.mplot3d import Axes3D  # برای رسم نمودار سه‌بعدی
```

حالا قصد داریم سه ویژگی دلخواه انتخاب کرده و پراکندگی نمونه ها را به صورت سه بعدی و با تفکیک نوع گل زنبق نمایش دهیم.

تا قسمتی که برای هر گونه یک رنگ اختصاص می‌هیم، مشابه قسمت ۱ است. اما این بار بر حسب سه ویژگی و به صورت سه بعدی قصد داریم نمودار را رسم کنیم. پس:

```
# ۴. ایجاد نمودار سه‌بعدی
fig = plt.figure(figsize=(10, 7))  # تنظیم اندازه نمودار
ax = fig.add_subplot(111, projection="3d")  # افزودن محور سه‌بعدی

# ۵. رسم نقاط برای هر کلاس به‌صورت جداگانه
for species, color in colors.items():
    subset = df[df["species"] == species]  # فیلتر کردن داده‌ها بر اساس نوع گل
    ax.scatter(
        subset["sepal length (cm)"],
        subset["sepal width (cm)"],
        subset["petal length (cm)"],
        label=species,
        color=color,
        s=60  # تنظیم اندازه نقاط
    )

# ۶. تنظیم برچسب محورها
ax.set_xlabel("Sepal Length (cm)")
ax.set_ylabel("Sepal Width (cm)")
ax.set_zlabel("Petal Length (cm)")
ax.set_title("3D Scatter Plot of Iris Dataset")

# ۷. نمایش راهنمای کلاس‌ها
ax.legend(title="Species")

# ۸. نمایش نمودار
plt.show()
```

در ادامه به توضیح خط به خط کد می‌پردازیم:

ایجاد نمودار سه‌بعدی

در این قسمت با استفاده از plt.figure(figsize=(10, 7)) اندازه کلی نمودار تعیین می‌شود که عرض ۱۰ اینچ و ارتفاع ۷ اینچ خواهد بود. سپس با fig.add_subplot(111, projection="3d") یک **محور سه‌بعدی** برای نمودار ایجاد می‌شود. این تنظیمات برای رسم نمودارهای سه‌بعدی ضروری است.

رسم نقاط برای هر کلاس به‌صورت جداگانه

در این بخش برای هر گونه از گل‌ها Setosa، Versicolorو Virginica ، داده‌های آن گونه فیلتر می‌شود و در نمودار سه‌بعدی به صورت نقاط رنگی نمایش داده می‌شود. از ax.scatter() برای رسم نقاط استفاده می‌شود که شامل ویژگی‌های **طول کاسبرگ، عرض کاسبرگ و طول گلبرگ** به ترتیب به عنوان محورهایX ، Yو Z است.

در اینجا

label=species به معنای نمایش نام گونه به عنوان برچسب در راهنمای نمودار است.

color=color رنگ مربوط به هر گونه را تعیین می‌کند.

s=60 اندازه نقاط را تنظیم می‌کند.

تنظیم برچسب محورها

در این بخش برچسب‌های محورهایX ، Yو Z تنظیم می‌شود

x.set_xlabel("Sepal Length (cm)") برچسب محور :X طول کاسبرگ

ax.set_ylabel("Sepal Width (cm)") برچسب محور :Y عرض کاسبرگ

ax.set_zlabel("Petal Length (cm)") برچسب محور :Z طول گلبرگ

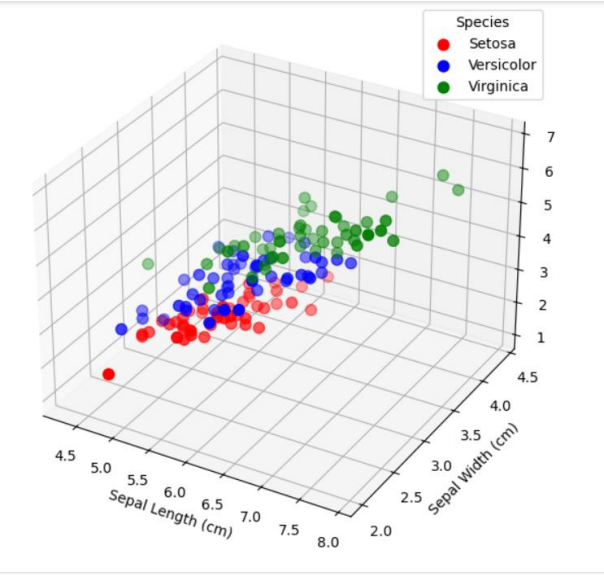
علاوه بر این، عنوان نمودار با ax.set_title("3D Scatter Plot of Iris Dataset") تنظیم می‌شود.

نمایش راهنمای کلاس‌ها

با استفاده از ax.legend(title="Species") یک راهنمای رنگ‌ها به نمودار اضافه می‌شود که نشان‌دهنده نوع گل‌ها و رنگ‌های مربوط به آن‌ها است.

نمایش نمودار

در نهایت، با استفاده از plt.show()، نمودار سه‌بعدی نمایش داده می‌شود که نقاط هر گونه را با رنگ‌های متفاوت نشان می‌دهد و ویژگی‌های مختلف گل زنبق را در سه بعد بررسی می‌کند. نتیجه این کد، یک نمودار سه‌بعدی است که به خوبی تفاوت ویژگی‌های طول کاسبرگ، عرض کاسبرگ و طول گلبرگ را برای سه گونه مختلف گل زنبق: Setosa ، Versicolorو Virginicaنمایش می‌دهد.



ب) ۳)

در این بخش قصد داریم heatmap داده‌ها را نمایش دهیم.

```
# ۱. بارگیری مجموعه داده Iris
iris = load_iris()

# ۲. تبدیل داده‌ها به DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# ۳. محاسبه‌ی ماتریس همبستگی (Correlation Matrix)
correlation_matrix = df.corr()

# ۴. رسم نقشه‌ی حرارتی با Seaborn
plt.figure(figsize=(8, 6))  # تنظیم اندازه نمودار
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)

# ۵. تنظیمات نمودار
plt.title("Heatmap of Feature Correlations in Iris Dataset")  #  عنوان نمودار

# ۶. نمایش نقشه‌ی حرارتی
plt.show()
```

ابتدا مثل قسمت‌های قبل مجموعه iris را بار گیری می‌کنیم و سپس داده‌های ویژگی‌های عددی از iris.data گرفته شده و به یک **DataFrame** در pandas تبدیل می‌شود.

محاسبه‌ی ماتریس همبستگی(Correlation Matrix)

correlation_matrix = df.corr()

تابع df.corr() میزان همبستگی بین ویژگی‌های مختلف را محاسبه می‌کند. مقدار همبستگی بین دو ویژگی عددی بین -۱ و +۱ قرار دارد

- مقدار ۱ نشان‌دهنده همبستگی مثبت کامل (یعنی وقتی یک ویژگی افزایش یابد، ویژگی دیگر نیز افزایش می‌یابد)
- مقدار ۰ یعنی هیچ رابطه‌ای بین دو ویژگی وجود ندارد
- مقدار -۱ نشان‌دهنده همبستگی منفی کامل (یعنی وقتی یک ویژگی افزایش یابد، ویژگی دیگر کاهش می‌یابد)

برای مثال، اگر مقدار همبستگی بین "petal length (cm)" و "petal width (cm)"برابر با ۰,۹۵ باشد، این یعنی این دو ویژگی رابطه‌ی بسیار قوی و مثبتی دارند.

چرا از ماتریس همبستگی استفاده کردیم:

میزان ارتباط بین ویژگی‌ها را مشخص می‌کند .این ماتریس نشان می‌دهد که چقدر دو ویژگی با هم رابطه دارند، با مقادیری بین **-۱ تا +۱**.

به ما کمک می‌کند الگوهای وابستگی را بهتر بفهمیم .اگر دو ویژگی همبستگی بالایی داشته باشند (مثلاً ۰,۹)، یعنی اطلاعات مشابهی دارند، اما اگر نزدیک به صفر باشد، یعنی ارتباط خاصی ندارند.

در یادگیری ماشین و تحلیل داده اهمیت دارد .اگر دو ویژگی خیلی به هم وابسته باشند، می‌توانیم یکی را حذف کنیم تا مدل پیچیدگی کمتری داشته باشد.

در نقشه‌ی حرارتی، یکی از رایج‌ترین روش‌ها برای نمایش روابط بین ویژگی‌هاست .چون به‌جای نمایش مستقیم داده‌ها، به ما تصویری از میزان ارتباط متغیرها می‌دهد که تحلیل را ساده‌تر می‌کند.

اگر به جای همبستگی، از خود داده‌های خام برای نقشه‌ی حرارتی استفاده کنیم، فقط **مقدار ویژگی‌ها را با رنگ‌های مختلف می‌بینیم**، اما نمی‌توانیم بفهمیم **کدام ویژگی‌ها به هم مرتبط هستند** .درحالی‌که نقشه‌ی حرارتی همبستگی دقیقاً این روابط را نشان می‌دهد.

رسم نقشه‌ی حرارتی باSeaborn :

plt.figure(figsize=(8, 6))

این دستور اندازه کلی نمودار را ۸ اینچ عرض و ۶ اینچ ارتفاع تعیین می‌کند

sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

این تابع از Seaborn برای رسم نقشه‌ی حرارتی (Heatmap) استفاده می‌شود که شدت همبستگی بین ویژگی‌ها را نشان می‌دهد.

correlation_matrix داده‌های همبستگی را به heatmap می‌دهد.

annot=True باعث می‌شود که مقدار عددی هر همبستگی روی خانه‌های نمودار نمایش داده شود.

cmap="coolwarm" رنگ‌بندی نمودار را مشخص می‌کند (رنگ‌های گرم‌تر نشان‌دهنده همبستگی مثبت و رنگ‌های سردتر نشان‌دهنده همبستگی منفی هستند)

fmt=".2f" تعیین می‌کند که مقدار همبستگی با دو رقم اعشار نمایش داده شود.

linewidths=0.5 فاصله‌ی بین خانه‌های نمودار را مشخص می‌کند تا خوانایی بهتری داشته باشد.

تنظیمات نمودار:

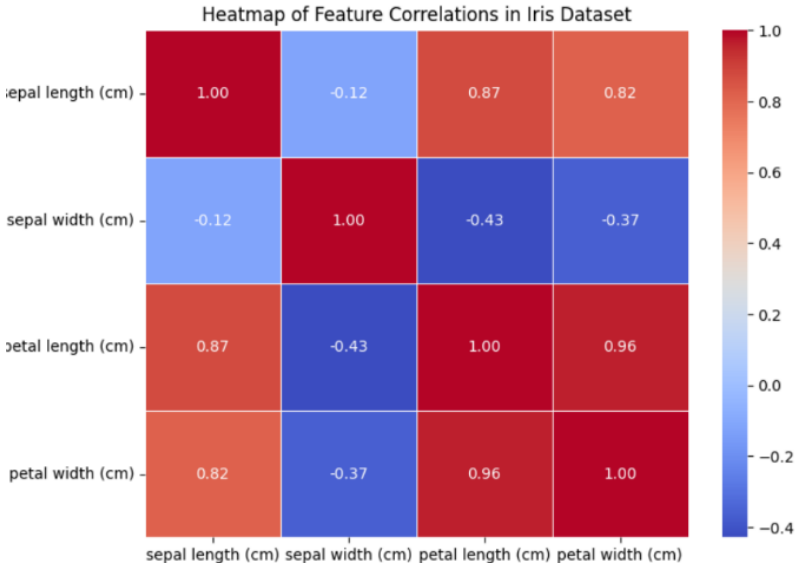
plt.title("Heatmap of Feature Correlations in Iris Dataset")

عنوان نمودار را تعیین می‌کند تا مشخص باشد که این نقشه‌ی حرارتی مربوط به کدام داده‌ها است.

نمایش نقشه‌ی حرارتی:

plt.show()

این دستور باعث می‌شود که نمودار نهایی روی صفحه نمایش داده شود.



نمودار Heatmap نمایش داده می‌شود که شدت همبستگی بین ویژگی‌های مختلف مجموعه داده Iris را نشان می‌دهد. رنگ‌های گرم‌تر (قرمز/نارنجی) نشان‌دهنده همبستگی مثبت قوی و رنگ‌های سردتر (آبی) نشان‌دهنده همبستگی منفی هستند. مقادیر عددی روی هر خانه نشان می‌دهد که مقدار دقیق همبستگی چقدر است.

ب) ۴) اینجا قصد داریم تابع چگالی احتمال ویژگی های دادگان را به تفکیک داده ی آموزش و آزمون نمایش دهیم.

```
# 1. Load the Iris dataset
iris = load_iris()

# 2. Convert the dataset to a Pandas DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["species"] = iris.target  # Add species column

# 3. Split the dataset into training (80%) and testing (20%) sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=63)
```

در ابتدا مجموعه داده با استفاده از load_iris() بارگیری شده و به یک DataFrame تبدیل می‌شود.

ستون species نیز به آن اضافه می‌شود که نشان می‌دهد هر نمونه به کدام کلاس تعلق دارد.

سپس داده‌ها با استفاده از train_test_split(df, test_size=0.2, random_state=63) تقسیم می‌شوند. ۸۰ درصد داده‌ها در مجموعه آموزشی (train_df) و ۲۰ درصد در مجموعه آزمایشی (test_df) قرار می‌گیرند. مقدار random_state=63 باعث می‌شود که این تقسیم‌بندی همیشه یکسان باشد.

```
plt.figure(figsize=(12, 8))
```

این خط اندازه کل نمودارها را تنظیم می‌کند figsize=(12,8). یعنی عرض ۱۲ واحد و ارتفاع ۸ واحد که باعث می‌شود نمودارها بهتر دیده شوند.

رسم نمودار چگالی احتمال (PDF) برای هر ویژگی:

```
for i, feature in enumerate(df.columns[:-1]):  # Exclude the 'species' column
    plt.subplot(2, 2, i + 1)  # Create subplots (2 rows, 2 columns)

    # Plot training data distribution
    sns.kdeplot(train_df[feature], label="Training Data", fill=True, alpha=0.6,
color="blue")

    # Plot testing data distribution
    sns.kdeplot(test_df[feature], label="Testing Data", fill=True, alpha=0.6,
color="red")
```

df.columns[:-1] یعنی چهار ویژگی اصلی را انتخاب کن و ستون "species" را نادیده بگیر چون ما فقط روی ویژگی‌های عددی تمرکز داریم.

plt.subplot(2, 2, i + 1) یعنی چهار نمودار جداگانه ایجاد کن که در یک صفحه با ۲ ردیف و ۲ ستون نمایش داده می‌شوند. حلقه for روی تمامی ویژگی‌های عددی مجموعه داده اجرا می‌شود و برای هر ویژگی یک نمودار جداگانه رسم می‌کند.

درون این حلقه ابتدا نمودار توزیع داده‌های آموزشی با sns.kdeplot(train_df[feature], label="Training Data", fill=True, alpha=0.6, color="blue") رسم می‌شود که به رنگ آبی نمایش داده می‌شود. سپس نمودار داده‌های تست با sns.kdeplot(test_df[feature], label="Testing Data", fill=True, alpha=0.6, color="red") رسم می‌شود که به رنگ قرمز نمایش داده می‌شود.

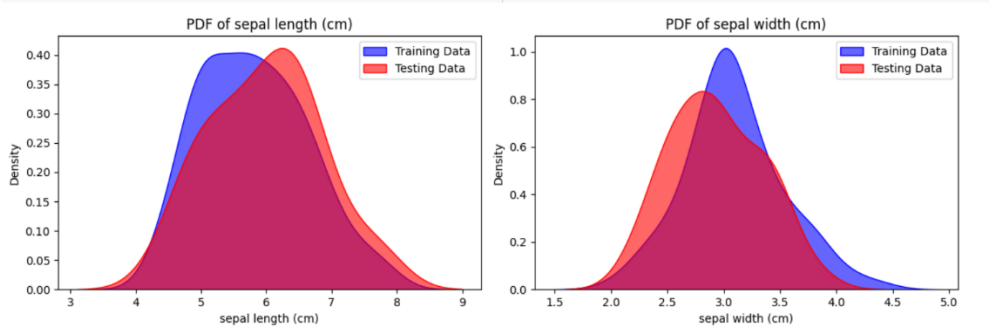
متغیر fill=True باعث می‌شود که نواحی زیر منحنی‌ها رنگی شوند تا دید بهتری ایجاد شود. مقدار alpha=0.6 شفافیت رنگ را مشخص می‌کند تا نمودارها روی یکدیگر قرار بگیرند و بتوان مقایسه‌ای بین آنها انجام داد.

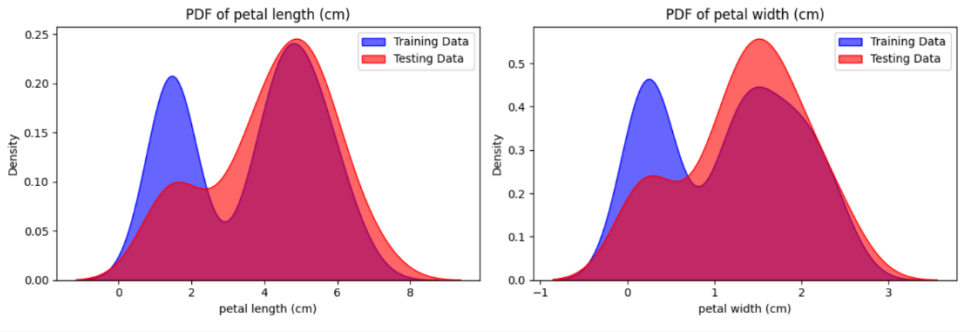
```
 # Set titles and labels
    plt.title(f"PDF of {feature}")
    plt.xlabel(feature)
    plt.ylabel("Density")
    plt.legend()

# 5. Display the plots
plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```

عنوان هر نمودار با plt.title(f"PDF of {feature}") تنظیم می‌شود. محور افقی مقدار ویژگی و محور عمودی مقدار چگالی احتمال را نشان می‌دهد plt.legend() .نیز راهنمای رنگ‌ها را نمایش می‌دهد تا مشخص شود کدام منحنی مربوط به داده‌های آموزشی و کدام یک مربوط به داده‌های آزمایشی است.

در نهایت plt.tight_layout() فاصله بین نمودارها را طوری تنظیم می‌کند که روی هم نیفتند وplt.show() نمودارها را نمایش می‌دهد.

خروجی در نهایت به این شکل است:





ج)

در این قسمت سعی داریم یکی از مقادیر پیوسته عددی را گرفته و به صورت گسسته کلاس بندی کنیم و سپس به عنوان یکی ویژگی جداگانه در DataFrame ذخیره کنیم

برای اینکار ما ابتدا نوع کلاس بندی را تعیین میکنیم و برای داده ها کلاس short , medium , Long انتخاب میکنیم

ابتدا داده های گل زنبق را لود میکنیم و مشابه موارد قبل به کمک دستور pd.DataFrame ذخیره میکنیم و نام ستون های ذخیره شده را همان نام ویژگی گل زنبق میگذاریم

iris = load_iris

df = pd.DataFrame(iris.data, columns=iris.feature_names)

سپس برای گسسته کردن داده ها از دستور pd.cut استفاده میکنیم و ویژگی مورد نظر را که طول برگ این مدل گل زنبق است را به سه قسمت تبدیل میکنیم و برای هر کدام به اندازه طولش یک لیبل مشخص میکنیم و سپس آن را نمایش میدهیم

پس کد به صورت زیر خواهد بود:

```
# 1. Load the Iris dataset
iris = load_iris()

# 2. Convert the dataset to a Pandas DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# 3. Select a numerical feature (sepal length) and discretize it into 3 categories
df["sepal_length_category"] = pd.cut(
    df["sepal length (cm)"],  # The continuous feature
    bins=3,  # Divide into 3 bins
    labels=["Short", "Medium", "Tall"]  # Assign labels to each bin
)

# 4. Display the first few rows to check the new column
print(df[["sepal length (cm)", "sepal_length_category"]].head())
```

د)

حال میخواهیم در این بخش به کمک دستور describe. ویژگی های آماری برای کلاس setosa بررسی کنیم

برای این کار باید داده های گل زنبق را لود کرده و همانند قسمت قبل در دیتا فریم ذخیره کنیم و به هر ستون نام ویژگی گل زنبق را نسبت دهیم

حال به کمک دستور iris.target کلاس های عددی 0,1,2 را به ستون نوع داده اضافه میکنیم و در نهایت به جای این اعداد از اسم استفاده میکنیم

df["species"] = iris.target

df["species"] = df["species"].map({0: "Setosa", 1: "Versicolor", 2: "Virginica"})

در نهایت با فیلتر کردن، کلاس Setosa را جدا کرده و به کمک دستور describe. این داده را تحلیل آماری میکنیم

setosa_df = df[df["species"] == "Setosa"]

setosa_stats = setosa_df.describe()

پس بنابراین کد نهایی به صورت زیر خواهد بود:

```
# 1. Load the Iris dataset
iris = load_iris()

# 2. Convert the dataset to a Pandas DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["species"] = iris.target  # Add species column
df["species"] = df["species"].map({0: "Setosa", 1: "Versicolor", 2: "Virginica"})  #
Map target values to class names

# 3. Filter only the Setosa class
setosa_df = df[df["species"] == "Setosa"]

# 4. Get statistical description of the Setosa class
setosa_stats = setosa_df.describe()

# 5. Display the result
print(setosa_stats
```