مخزن گیت‌هاب

برای مشاهده و دانلود کدهای مربوط به این پروژه، می‌توانید به مخزن گیت‌هاب مراجعه کنید:

مخزن گیت‌هاب پروژه‌ها

دفترچه گوگل کولب

همچنین دفترچه‌ی گوگل کولب مربوط به تمرین را از لینک زیر مشاهده کنید:

AI۴۰۳۲_HW۱ در Colab Google

گزارش تمرین ۱
هوش مصنوعی

دکتر علیاری

سینا حسن‌پور   شماره دانشجویی: ۴۰۲۱۶۷۲۳

۱۸ اسفند ۱۴۰۳

۱ مقداری جبر خطی

I ضرب و ابعاد

ماتریس‌های زیر را در نظر بگیرید:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}$$

آ) طبق قاعده‌ی ضرب ماتریس‌ها:

اگه ماتریس A ابعاد $m \times n$ داشته باشه و ماتریس B ابعاد $n \times p$ داشته باشه، می‌تونیم AB رو حساب کنیم و ماتریس حاصل ابعاد $m \times p$ داره.
اگه تعداد ستون‌های A با تعداد سطرهای B برابر نباشه، نمی‌تونیم ضرب رو انجام بدیم!
ابعاد این ماتریس‌ها:

- ابعاد A : 2×3
- ابعاد B : 4×2
- ابعاد B^T (ترانهاده‌ی B) : 2×4
- ابعاد A^T (ترانهاده‌ی A) : 3×2

بررسی امکان‌پذیری ضرب‌ها با بررسی ابعاد، می‌توان دریافت که: - ضرب BA امکان‌پذیر است و نتیجه‌ی آن یک ماتریس با ابعاد 4×3 خواهد بود. - ترانهاده‌ی B، یعنی B^T، دارای ابعاد 2×4 است. - ضرب‌های $B^T A$ و $A^T B$ به دلیل اینکه برخلاف قاعده ضرب بودند، تعریف نشده‌اند .

ب) محاسبه‌ی BA و B^T

محاسبه‌ی B^T

ترانهاده‌ی ماتریس B به‌صورت زیر است:

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

محاسبه‌ی BA

عملیات ضرب ماتریسی برای BA انجام می‌شود:

$$BA = \begin{bmatrix} (b_{11}a_{11} + b_{12}a_{21}) & (b_{11}a_{12} + b_{12}a_{22}) & (b_{11}a_{13} + b_{12}a_{23}) \\ (b_{21}a_{11} + b_{22}a_{21}) & (b_{21}a_{12} + b_{22}a_{22}) & (b_{21}a_{13} + b_{22}a_{23}) \\ (b_{31}a_{11} + b_{32}a_{21}) & (b_{31}a_{12} + b_{32}a_{22}) & (b_{31}a_{13} + b_{32}a_{23}) \\ (b_{41}a_{11} + b_{42}a_{21}) & (b_{41}a_{12} + b_{42}a_{22}) & (b_{41}a_{13} + b_{42}a_{23}) \end{bmatrix}$$

II ضرب و ابعاد، دشوارتر

پاسخ به سؤالات (الف) و (ب)

الف) بررسی ابعاد

- فرض کنید $x^{(i)} \in \mathbb{R}^{1\times 2}$، یعنی $x^{(i)} = \left[x_1^{(i)},\ x_2^{(i)}\right]$.
- اگر $\theta = \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix} \in \mathbb{R}^{2\times 1}$, آنگاه ضرب $x^{(i)}\,\theta$ که ابعاد $(1 \times 2) \times (2 \times 1)$ دارد، یک اسکالر (1×1) خواهد بود:
$$x^{(i)}\,\theta \;=\; [\,x_1^{(i)},\ x_2^{(i)}\,]\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix} \;=\; x_1^{(i)}\,\theta_1 \;+\; x_2^{(i)}\,\theta_2 \;\in\; \mathbb{R}.$$
- بعد θ در این مثال (2×1) است و بنابراین بعد $x^{(i)}\,\theta$ یا $\theta^T x^{(i)}$ اسکالر (1×1) است.
- وقتی همه‌ی نمونه‌ها $x^{(i)} \in \mathbb{R}^{1\times 2}$ (برای $i = 1, \ldots, n$) را در یک ماتریس جمع کنیم، ماتریس
$$X \;\in\; \mathbb{R}^{n\times 2}$$
به دست می‌آید که هر سطر آن یکی از بردارهای $x^{(i)}$ است:
$$X \;=\; \begin{bmatrix} -(x^{(1)})- \\ -(x^{(2)})- \\ \vdots \\ -(x^{(n)})- \end{bmatrix} \;=\; \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} \end{bmatrix}.$$
بعد این ماتریس $n \times 2$ است.
- ضرب $X\,\theta$ آنگاه ابعاد $n \times 1$ خواهد داشت:
$$X\,\theta \;\in\; \mathbb{R}^{n\times 1}.$$

ب) بازنویسی تابع هزینه و محاسبه‌ی گرادیان تابع هزینه (Cost Function) رگرسیون خطی معمولاً به صورت

$$J(\theta) \;=\; \sum_{i=1}^{n} \Bigl(x^{(i)}\,\theta \;-\; y^{(i)}\Bigr)^2$$

تعریف می‌شود.

- **نوشتن در قالب ضرب برداری:**
اگر $X \in \mathbb{R}^{n\times 2}, \quad \theta \in \mathbb{R}^{2\times 1}, \quad y \in \mathbb{R}^{n\times 1},$ آنگاه

$$J(\theta) \;=\; (X\,\theta - y)^T\,(X\,\theta - y).$$

این همان جمع مربعات مؤلفه‌های بردار $(X\,\theta - y)$ است، یعنی:

$$(X\,\theta - y)^T(X\,\theta - y) \;=\; \sum_{i=1}^{n} \Bigl([x^{(i)}\,\theta] - y^{(i)}\Bigr)^2.$$

- **بسط و محاسبه‌ی گرادیان:**

$$J(\theta) \;=\; (X\,\theta - y)^T\,(X\,\theta - y) \;=\; \theta^T X^T X\,\theta \;-\; 2\,y^T X\,\theta \;+\; y^T y.$$

مشتق این عبارت نسبت به θ (یا $\nabla_\theta J(\theta)$) برابر است با:

$$\nabla_\theta J(\theta) \;=\; 2\,X^T\,\bigl(X\,\theta - y\bigr).$$

(اگر در تعریف اولیه‌ی $J(\theta)$ ضریب $\frac{1}{2}$ به کار رفته باشد، ضریب ۲ در گرادیان حذف می‌گردد.)

نتیجه‌گیری نهایی

- بعد θ در این مثال (2×1) است.
- بعد $x^{(i)}$ (1×2) و بنابراین $x^{(i)}\,\theta$ اسکالر (1×1).
- با کنار هم گذاشتن تمامی $x^{(i)}$ها، ماتریس X با بعد $(n \times 2)$ ساخته می‌شود و ضرب $X\,\theta$ در ابعاد $n \times 1$ خواهد بود.
- تابع هزینه $J(\theta)$ را می‌توان به‌صورت $(X\theta - y)^T(X\theta - y)$ نوشت و گرادیان آن نسبت به θ نیز $2\,X^T(X\theta - y)$ به دست می‌آید.

۲ پردازش داده

Dataset CWRU I

آ) دریافت داده :

فرمت فایل داده شده ‘.mat‘ است که در نرم‌افزار MATLAB استفاده میشود . برای خواندن این نوع فایل در پایتون، از کتابخانه ‘scipy.io‘ استفاده می‌کنیم. دستور زیر:
در اینجا کد به زبان پایتون آورده شده است:

```
data = scipy.io.loadmat("/content/252.mat")
print("Keys in MAT file:", data.keys())
```

خروجی:

```
Keys in MAT file: dict_keys(['__header__', '__version__', '__globals__',
'X252_DE_time', 'X252_FE_time', 'X252RPM'])
```

در اینجا، فایل ۲۵۲.MAT خوانده می‌شود و در متغیر data ذخیره می‌شود. با data.keys() می‌توانیم کلیدهای موجود در فایل را مشاهده کنیم.

پس اجزای آن شامل موارد زیر است:

header ← اطلاعات مربوط به فرمت فایل .MAT
version ← نسخه‌ی فایل .MAT
globals ← متغیرهای عمومی در فایل
X۲۵۲_DE_time ← سیگنال لرزش انتهای درایو
X۲۵۲_FE_time ← سیگنال لرزش انتهای فن
X۲۵۲RPM ← سرعت چرخش (RPM)

بررسی نوع داده خوانده شده با دستور زیر:

```
print(type(data))
```

خروجی:

```
<class 'dict'>
```

نتیجه: داده‌ی خوانده شده از نوع دیکشنری (dict) است.

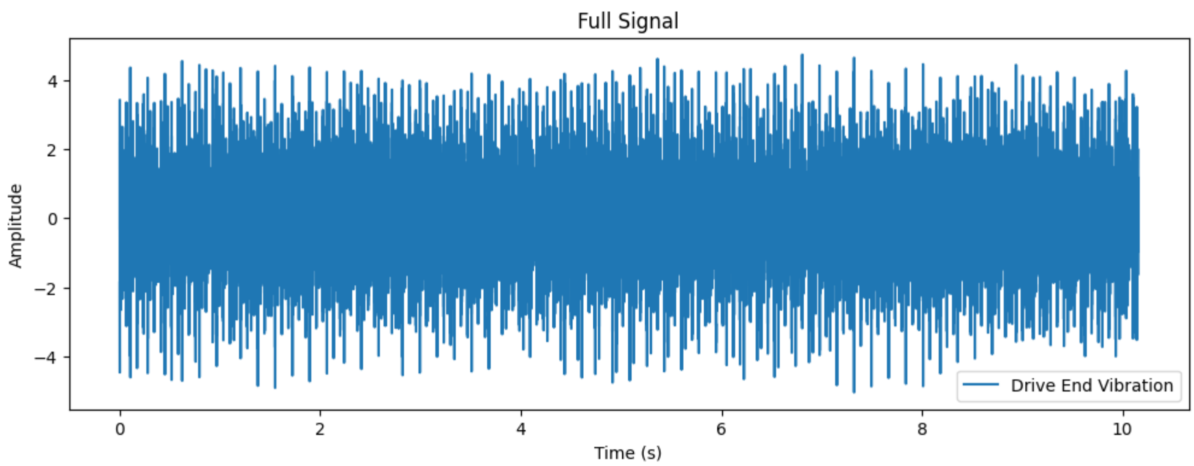
از بین سیگنال‌های موجود، می‌توانیم یکی از ستون‌های X۲۵۲_DE_time یا X۲۵۲_FE_time را انتخاب کنیم. در اینجا، سیگنال X۲۵۲_DE_time را انتخاب می‌کنیم. انتخاب سیگنال لرزش انتهای درایو و ذخیره در متغیر:

```
signal = data["X252_DE_time"].flatten()
```

ب) نمایش سیگنال :

فرکانس نمونه‌برداری (Sampling Frequency): گفته شده که ۴۸KHz است. یعنی هر ثانیه ۴۸۰۰۰ نمونه ثبت شده است.
پس اگر N نمونه داشته باشیم، زمان معادل آن را می‌توان به این شکل محاسبه کرد:
index fs time= ⍰ ⍰ index = time
کد برای رسم کل سیگنال:

```
fs = 48000
N = len(signal)

time = np.arange(N) / fs

plt.figure(figsize=(12, 4))
plt.plot(time, signal, label="Drive End Vibration")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("Full Signal")
plt.legend()
plt.show()
```

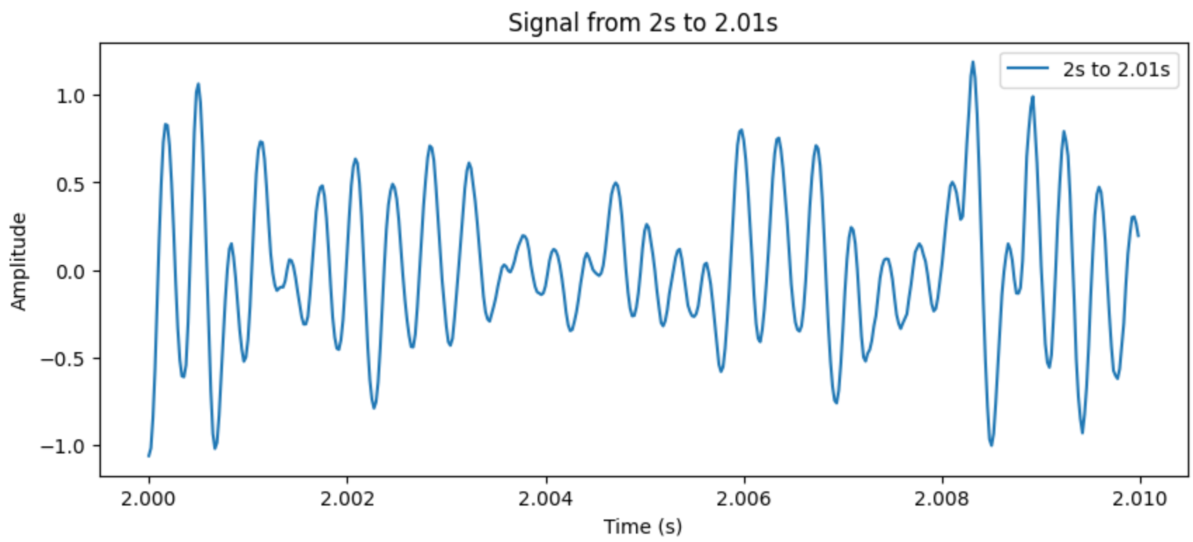


نمایش سیگنال در بازه‌ی $2s$ تا $2.01s$
محاسبه‌ی تعداد نمونه‌ها در این بازه: بازه‌ی $2s$ تا $2.01s$ یعنی ۱۰ میلی‌ثانیه که معادل:

$$\text{Number of samples} = 0.01 \times 48000 = 480 \text{ samples}$$

کد برای رسم این بخش:

```
start_idx = int(2 * fs)  #start
end_idx = start_idx + 480  #end sample

plt.figure(figsize=(10, 4))
plt.plot(time[start_idx:end_idx], signal[start_idx:end_idx], label="2s to
    2.01s")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("Signal from 2s to 2.01s")
plt.legend()
plt.show()
```

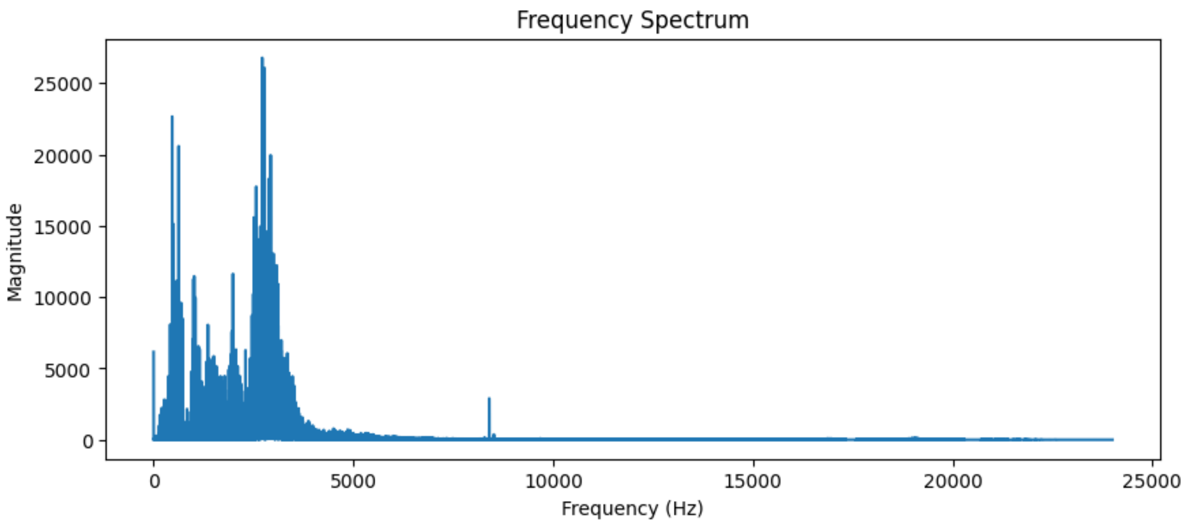


ج) تحلیل فرکانسی :

تبدیل فوریه (FFT) برای پیدا کردن فرکانس‌های موجود در سیگنال انجام می‌شود.
مراحل:
تبدیل سیگنال به حوزه‌ی فرکانس با np.fft.fft()
محور فرکانس را با np.fft.fftfreq() ایجاد کنیم
قدرتمندی طیف را با np.abs() محاسبه کنیم
کد:

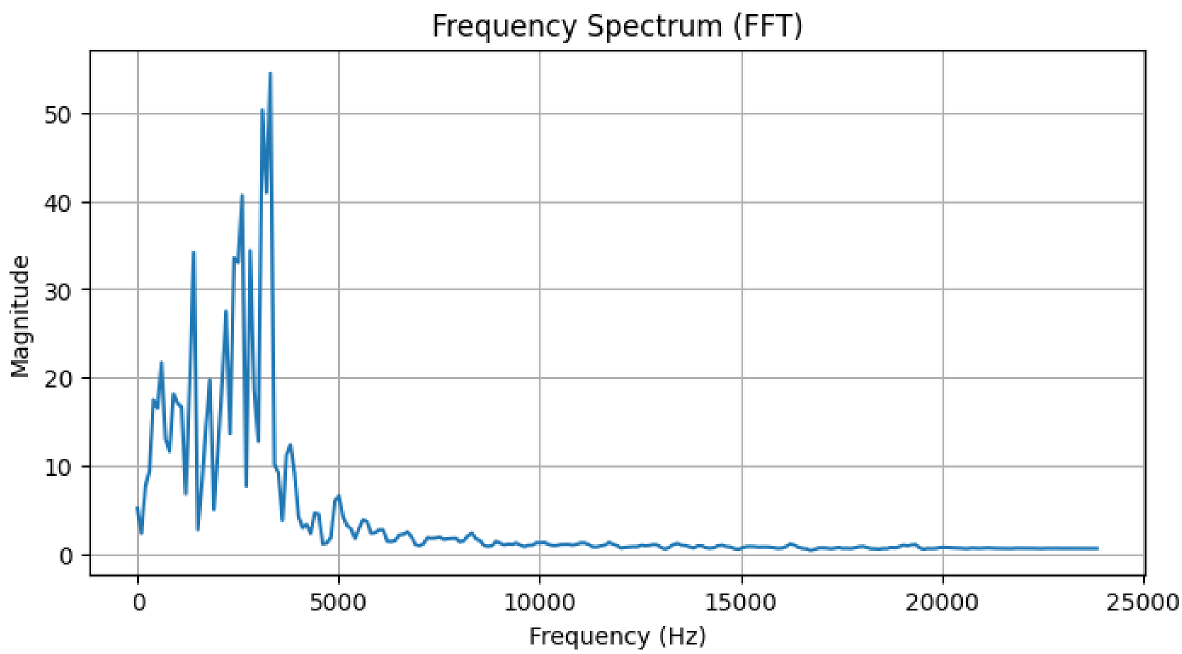
```
fft_vals = np.fft.fft(signal)
freqs = np.fft.fftfreq(len(signal), d=1/fs)

magnitude = np.abs(fft_vals)

plt.figure(figsize=(10, 4))
plt.plot(freqs[:N//2], magnitude[:N//2])
plt.xlabel("Frequency (Hz)")
plt.ylabel("Magnitude")
plt.title("Frequency Spectrum")
plt.show()
```

۱. رسم طیف فرکانسی



حال یک تابع می‌نویسم که: یک بازه زمانی دلخواه از سیگنال را به عنوان ورودی بگیرد، تبدیل فوریه (FFT) را روی آن اجرا کندو طیف فرکانسی را نمایش دهد.
کد تابع برای نمایش طیف فرکانسی با تبدیل فوریه:

```python
selected_signal = data['X252_DE_time'].squeeze()
def plot_frequency_spectrum(signal, sample_rate, start_time=0,
    end_time=None):

    total_samples = len(signal)

    if end_time is None:
        end_time = total_samples / sample_rate

    start_index = int(start_time * sample_rate)
    end_index = int(end_time * sample_rate)

    selected_signal = signal[start_index:end_index]

    fft_result = np.fft.fft(selected_signal)
    freq = np.fft.fftfreq(len(selected_signal), d=1/sample_rate)

    positive_freqs = freq[:len(freq)//2]
    magnitude = np.abs(fft_result[:len(freq)//2])

    plt.figure(figsize=(8,4))
    plt.plot(positive_freqs, magnitude)
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.title("Frequency Spectrum (FFT)")
    plt.grid()
    plt.show()

plot_frequency_spectrum(selected_signal, sample_rate=48000, start_time=2,
    end_time=2.01)
```



squeeze() باعث می‌شود که اگر داده‌های ما به صورت ماتریس ذخیره شده باشند، به یک آرایه یک‌بعدی تبدیل شوند که پردازش آن آسان‌تر است. برای نمایش طیف فرکانسی سیگنال، یک تابع عمومی نوشته‌ایم که می‌تواند هر سیگنال را در یک بازه‌ی مشخص دریافت کرده و تبدیل فوریه‌ی آن را محاسبه کند.
np.fft.fft() تبدیل فوریه را روی سیگنال اعمال می‌کند.
np.fft.fftfreq() فرکانس‌های متناظر با خروجی تبدیل فوریه را محاسبه می‌کند.
فقط نیمه‌ی مثبت فرکانس‌ها نمایش داده می‌شود، زیرا در تبدیل فوریه، قسمت منفی فرکانس‌ها تقارن دارد و اطلاعات اضافه‌ای ندارد.
نمودار خروجی با محور افقی فرکانس (Hz) و محور عمودی دامنه (Magnitude) رسم می‌شود.

۲. پیدا کردن فرکانس غالب (بیشترین انرژی در طیف)

```
dominant_freq = freqs[np.argmax(magnitude[:N//2])]
print("Dominant Frequency:", dominant_freq, "Hz")
```

خروجی:

Dominant Frequency: 2719.5640398535857 Hz

د ، ه) تقسیم بندی سیگنال

مراحل:
سیگنال را به قطعات ۱۲۸ تایی بشکنیم.
هر قطعه را در یک سطر از numpy array ذخیره کنیم.
مقدار هیپوتنوسی (Norm) هر قطعه را حساب کنیم.

```
import numpy as np
segment_size = 128
num_segments = len(signal) // segment_size

segments = signal[:num_segments * segment_size].reshape(num_segments,
    segment_size)

segment_norms = np.linalg.norm(segments, axis=1)

print("Number of pieces:", num_segments)
print("Some sample hypotenuse values:", segment_norms[:5])
```

تعداد قطعات: ۳۸۰۷
چند مقدار هیپوتنوسی نمونه:

[18.75708897 9.5150878 12.12775435 12.15589371 8.60514374]

محاسبه‌ی مقدار هیپوتنوسی هر قطعه و ذخیره آن
(پاسخ به: "مقدار هیپوتنوسی قطعه‌ها را ذخیره کنید.")

```
segment_norms = np.linalg.norm(segments, axis=1)
print("First 10 segment norms:", segment_norms[:10])
```

خروجی:

First 10 segment norms: [18.75708897 9.5150878 12.12775435 12.15589371 8.60514374 6.47621605 3.01400884 4.63807857 3.21532278 1.87234841]

ذخیره قطعات در DataFrame

مراحل: هر قطعه را در یک pandas.DataFrame ذخیره کنیم. مقدار هیپوتنوسی هر قطعه را به آن اضافه کنیم.

```
import pandas as pd
df_segments = pd.DataFrame(segments)

df_segments["Norm"] = segment_norms

print(df_segments.head())
```

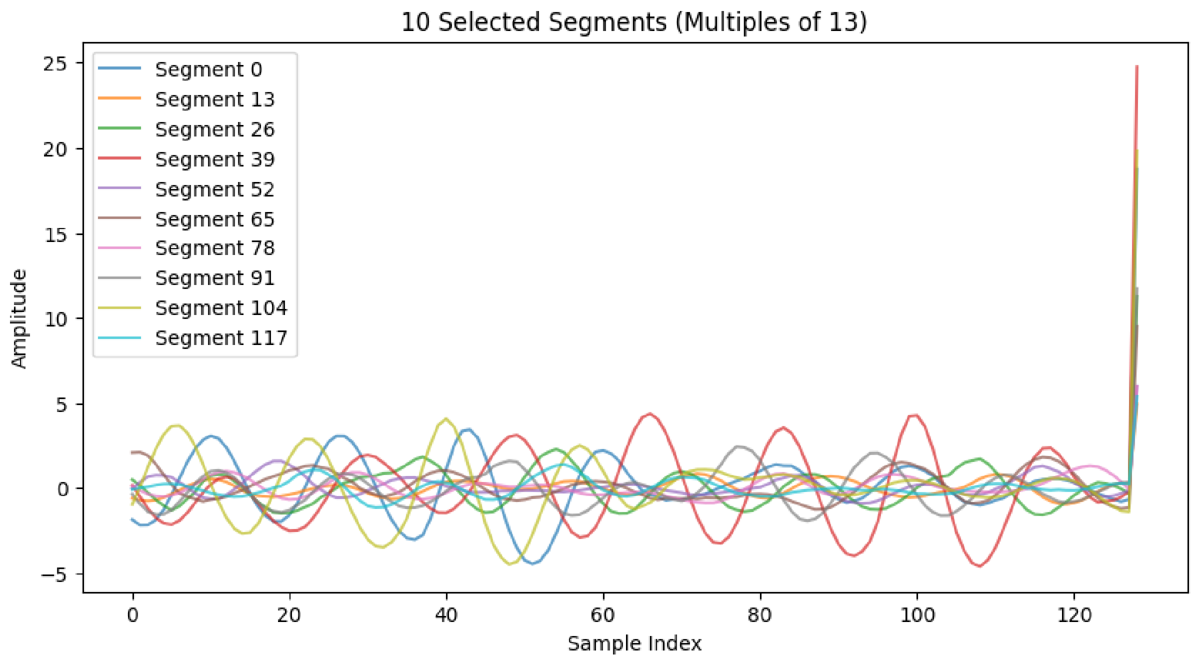
نمایش ۱۰ قطعه‌ی مضرب ۱۳ در یک نمودار

با استفاده از DataFrame ایجادشده، ۱۰ قطعه‌ی مضرب ۱۳ را در یک نمودار روی هم نمایش دهید. نمودارها باید رنگ‌های مختلف داشته باشند و نمودار نهایی نیز برچسب‌دار باشد. (تمامی این مراحل باید در یک حلقهٔ for انجام شود.)

توضیح:

۱۰ قطعه‌ی اولی که ایندکس آنها مضرب ۱۳ است را پیدا می‌کنیم. هرکدام را روی نمودار رسم می‌کنیم و از رنگ‌های متفاوت استفاده می‌کنیم.

```python
import matplotlib.pyplot as plt
selected_segments = df_segments.iloc[::13][:10].values

plt.figure(figsize=(10, 5))
for i, segment in enumerate(selected_segments):
    plt.plot(segment, label=f"Segment {i*13}", alpha=0.7)

plt.xlabel("Sample Index")
plt.ylabel("Amplitude")
plt.title("10 Selected Segments (Multiples of 13)")
plt.legend()
plt.show()
```



و) استخراج ویژگی (Feature Extraction)

```python
features = [extract_features(segment) for segment in segments]

df_features = pd.DataFrame(features)

print(df_features.head())
df_features.to_csv("extracted_features.csv", index=False)
print("File saved: extracted_features.csv")
from google.colab import files

files.download("extracted_features.csv")
```

- **تقسیم سیگنال به قطعات ۱۲۸ تایی** و ذخیره در `numpy`
- **ذخیره‌ی قطعات** در `pandas.DataFrame`
- **رسم ۱۰ قطعه‌ی مضرب ۱۳** روی یک نمودار
- **نوشتن تابعی برای استخراج ویژگی‌های آماری**
- **ذخیره‌ی ویژگی‌های محاسبه‌شده** در `DataFrame` و خروجی `CSV`

Iris Dataset II

مقدمه

تحقیق درباره دیتاست معروف Iris
دیتاست Iris که اولین بار توسط Sir R.A. Fisher استفاده شد، از مقاله‌ی فیشر گرفته شده است. این مجموعه داده همان مجموعه‌ای است که در R استفاده شده، اما کمی با نسخه‌ای که در UCI Machine Learning Repository وجود دارد، متفاوت است؛ زیرا نسخه‌ی UCI دو نقطه داده‌ی اشتباه دارد.
این مجموعه داده یکی از شناخته‌شده‌ترین دیتاست‌ها در ادبیات **شناسایی الگو** (Pattern Recognition) است. مقاله‌ی فیشر یک مقاله‌ی کلاسیک در این حوزه محسوب می‌شود و هنوز هم به آن ارجاع داده می‌شود (برای مثال، در کتاب Duda & Hart).

در این گزارش، به بررسی **Iris Dataset** پرداخته می‌شود که یکی از معروف‌ترین مجموعه‌داده‌ها در زمینه **یادگیری ماشین** و **تحلیل داده** است. این مجموعه داده شامل ویژگی‌های گل‌های **Iris** در سه گونه‌ی مختلف است و برای طبقه‌بندی استفاده می‌شود. هدف اصلی این بررسی، درک توزیع داده‌ها، تحلیل ویژگی‌ها و بررسی امکان‌پذیری طبقه‌بندی آنها بر اساس ویژگی‌هایشان است.

شرح داده‌ها

مجموعه داده‌ی **Iris** شامل **۱۵۰ نمونه از سه گونه‌ی گل Iris** است که به طور مساوی (۵۰ نمونه برای هر گونه) توزیع شده‌اند. هر نمونه دارای **۴ ویژگی عددی** است:

- **طول کاسبرگ** (Sepal Length)
- **عرض کاسبرگ** (Sepal Width)
- **طول گلبرگ** (Petal Length)
- **عرض گلبرگ** (Petal Width)

علاوه بر این ویژگی‌ها، یک **برچسب دسته‌ای (Class Label)** وجود دارد که نشان می‌دهد هر نمونه متعلق به کدام گونه از گل‌های **Iris** است.

- **Setosa**
- **Versicolor**
- **Virginica**

تحلیل آماری و بررسی ویژگی‌ها

توزیع داده‌ها

با بررسی داده‌ها، مشخص می‌شود که ویژگی‌های مختلف گل‌ها دارای **توزیع‌های متفاوتی** هستند. برخی ویژگی‌ها مانند **طول گلبرگ (Petal Length)** در گونه‌های مختلف تفاوت‌های واضحی دارند، در حالی که برخی ویژگی‌های دیگر مانند **عرض کاسبرگ (Sepal Width)** ممکن است هم‌پوشانی بیشتری بین گونه‌ها داشته باشند.

نمایش گرافیکی داده‌ها

برای درک بهتر داده‌ها، می‌توان از **نمودارهای توزیع، هیستوگرام، جعبه‌ای (Boxplot) و** scatter plot استفاده کرد.
به عنوان مثال:

- **نمودار پراکندگی (scatter plot)** نشان می‌دهد که داده‌های گونه‌ی **Setosa** به خوبی از دو گونه‌ی دیگر جدا شده‌اند.
- **نمودار هیستوگرام** به ما کمک می‌کند توزیع هر ویژگی را به صورت جداگانه ببینیم.
- **نمودار جعبه‌ای (Boxplot)** میزان پراکندگی و نقاط پرت (Outliers) را نشان می‌دهد.

تحلیل تفکیک‌پذیری کلاس‌ها

یکی از مهم‌ترین سوالات در این تحلیل این است که **آیا می‌توان گونه‌های مختلف را بر اساس ویژگی‌هایشان به درستی از هم تفکیک کرد؟**

- **Setosa به وضوح از بقیه گونه‌ها متمایز است.**
- **Versicolor و Virginica همپوشانی بیشتری دارند، ولی در برخی ویژگی‌ها مانند طول و عرض گلبرگ تفاوت مشخصی دارند.**

نتیجه‌گیری

مجموعه داده‌ی Iris یک مثال عالی برای تحلیل داده و **یادگیری ماشین** است. بررسی ویژگی‌های این داده‌ها نشان می‌دهد که **گونه‌های مختلف گل‌ها را می‌توان با تحلیل ویژگی‌هایشان از هم تفکیک کرد.** این مجموعه داده به دلیل **ساختار ساده ولی کاربردی‌اش**، یکی از اولین گزینه‌ها برای یادگیری مفاهیم **داده‌کاوی، طبقه‌بندی و تحلیل آماری** محسوب می‌شود.

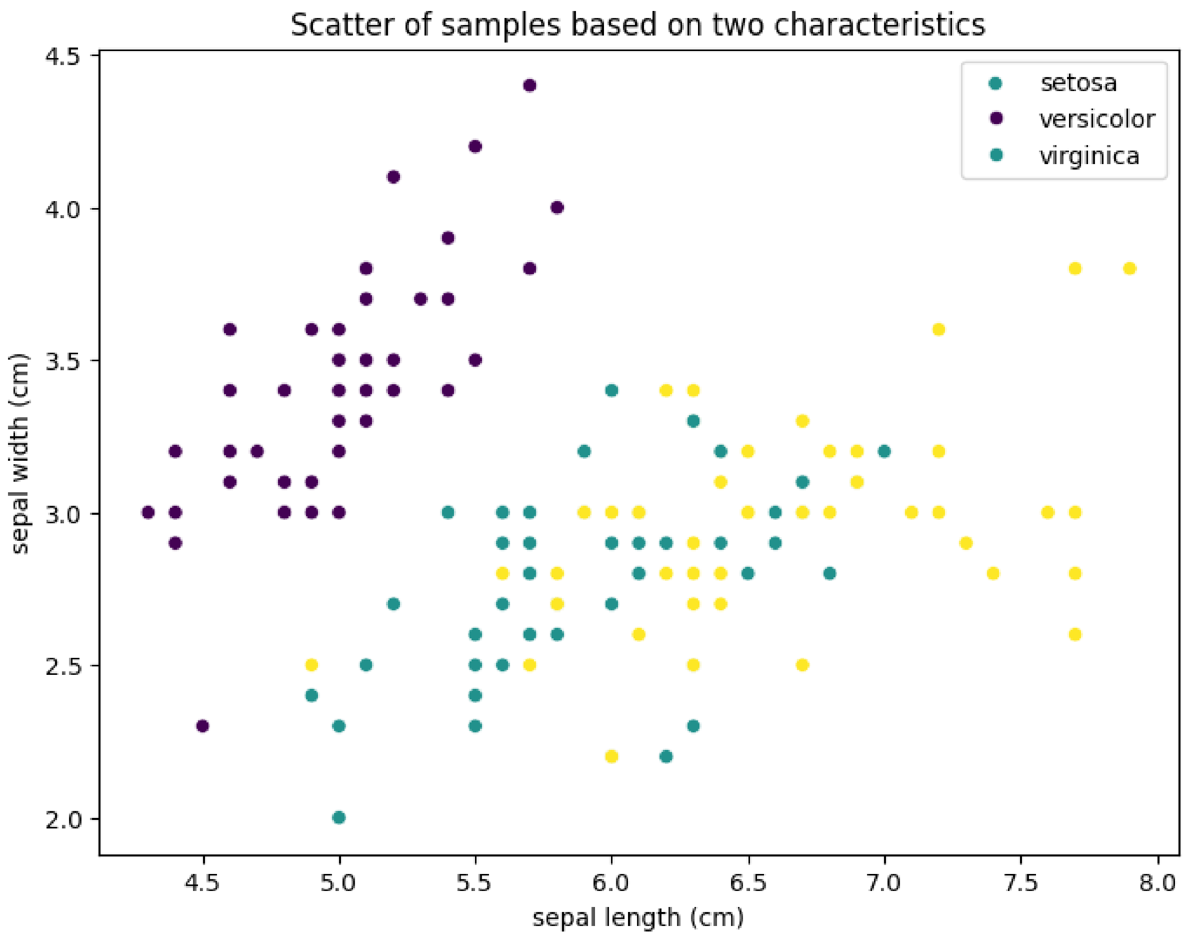
خلاصه‌ی کلی گزارش این بخش

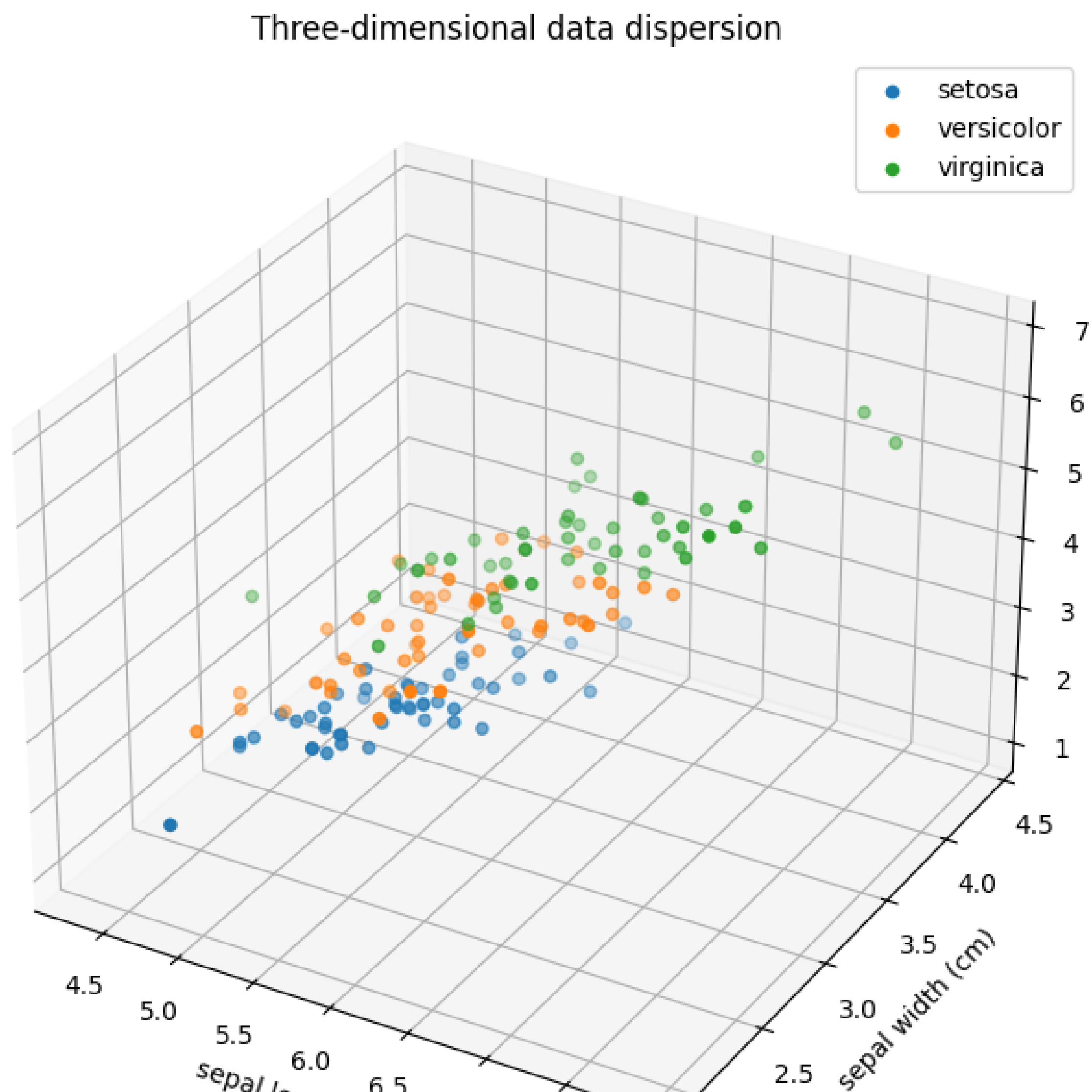
- **معرفی داده‌ها و ویژگی‌های آنها**
- **تحلیل توزیع داده‌ها و نمایش بصری**
- **بررسی امکان تفکیک‌پذیری کلاس‌ها**
- **نتیجه‌گیری درباره‌ی استفاده‌ی این داده‌ها در یادگیری ماشین**

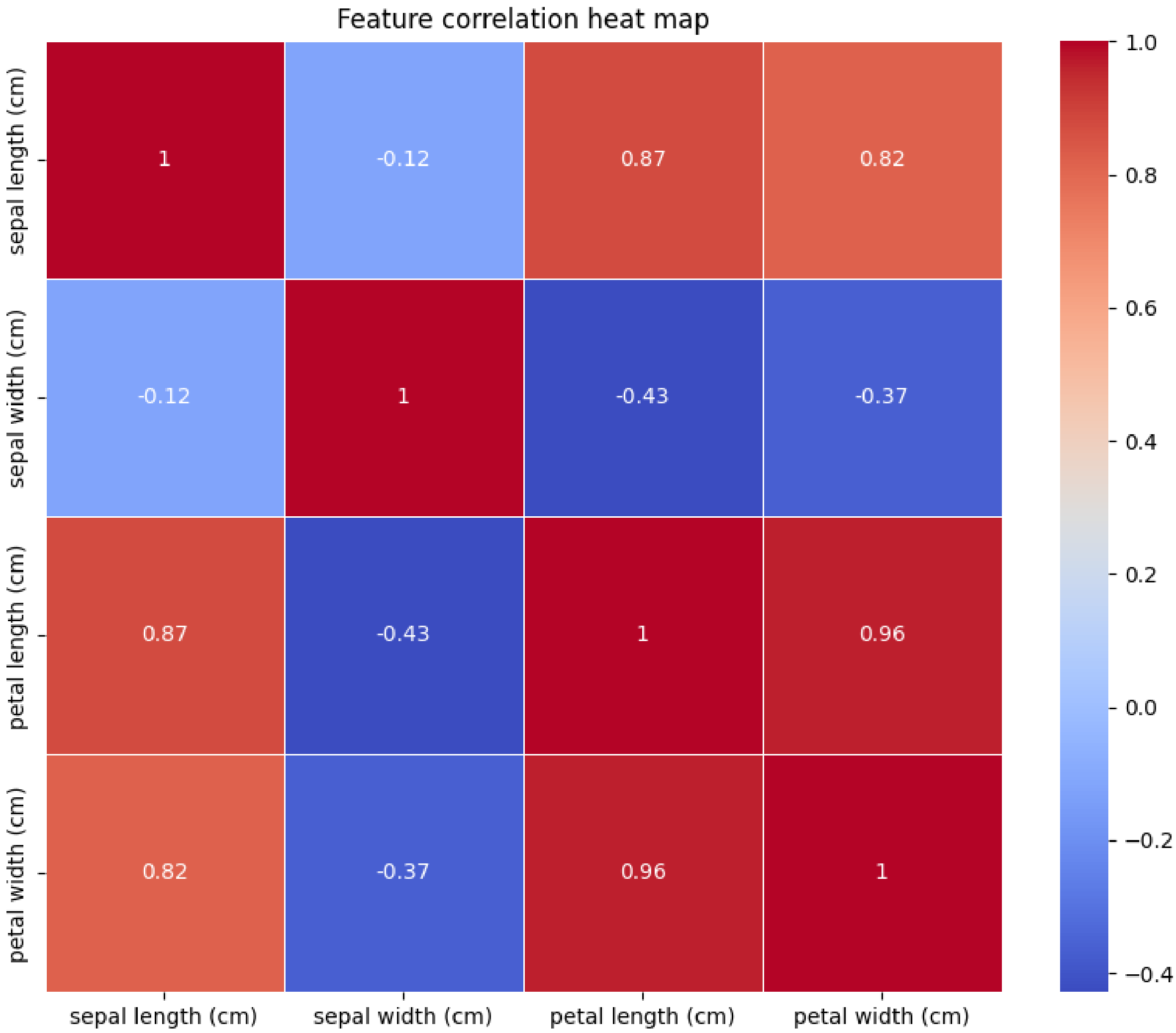
```python
# Import necessary libraries
from sklearn import datasets  # Load built-in datasets from scikit-learn
import pandas as pd  # Data manipulation and analysis
import numpy as np  # Numerical computations
from sklearn.model_selection import train_test_split  # Splitting dataset into train and test
import matplotlib.pyplot as plt  # Data visualization
import seaborn as sns  # Advanced visualization
from mpl_toolkits.mplot3d import Axes3D  # 3D plotting

# Load the Iris dataset
iris = datasets.load_iris()

# Print dataset description
print(iris.DESCR)

# Convert dataset to a Pandas DataFrame with feature names as columns
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
# Add the target variable (species label) to the DataFrame
data['target'] = iris.target

# Split the dataset into training (70%) and testing (30%) sets
data_train, data_test = train_test_split(data, test_size=0.3, random_state=42)

# Label the sets to differentiate between train and test data
data_train['set'] = 'train'
data_test['set'] = 'test'

# Combine training and test datasets into a single DataFrame
data_combined = pd.concat([data_train, data_test])

# Display the first five rows of the combined dataset
print(data_combined.head())

# 2D Scatter plot using the first two features
plt.figure(figsize=(8, 6))
sns.scatterplot(
    x=data_combined[iris.feature_names[0]],  # Sepal length
    y=data_combined[iris.feature_names[1]],  # Sepal width
    hue=data_combined['target'],  # Color by species
    palette='viridis'  # Color theme
)
plt.xlabel(iris.feature_names[0])  # Label x-axis
plt.ylabel(iris.feature_names[1])  # Label y-axis
plt.title("Scatter of samples based on two characteristics")  # Title of the plot
plt.legend(iris.target_names)  # Add species labels as legend
plt.show()

# 3D Scatter plot using the first three features
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Loop through each species and plot them in 3D
for i, species in enumerate(iris.target_names):
    subset = data_combined[data_combined['target'] == i]
    ax.scatter(subset[iris.feature_names[0]],
```

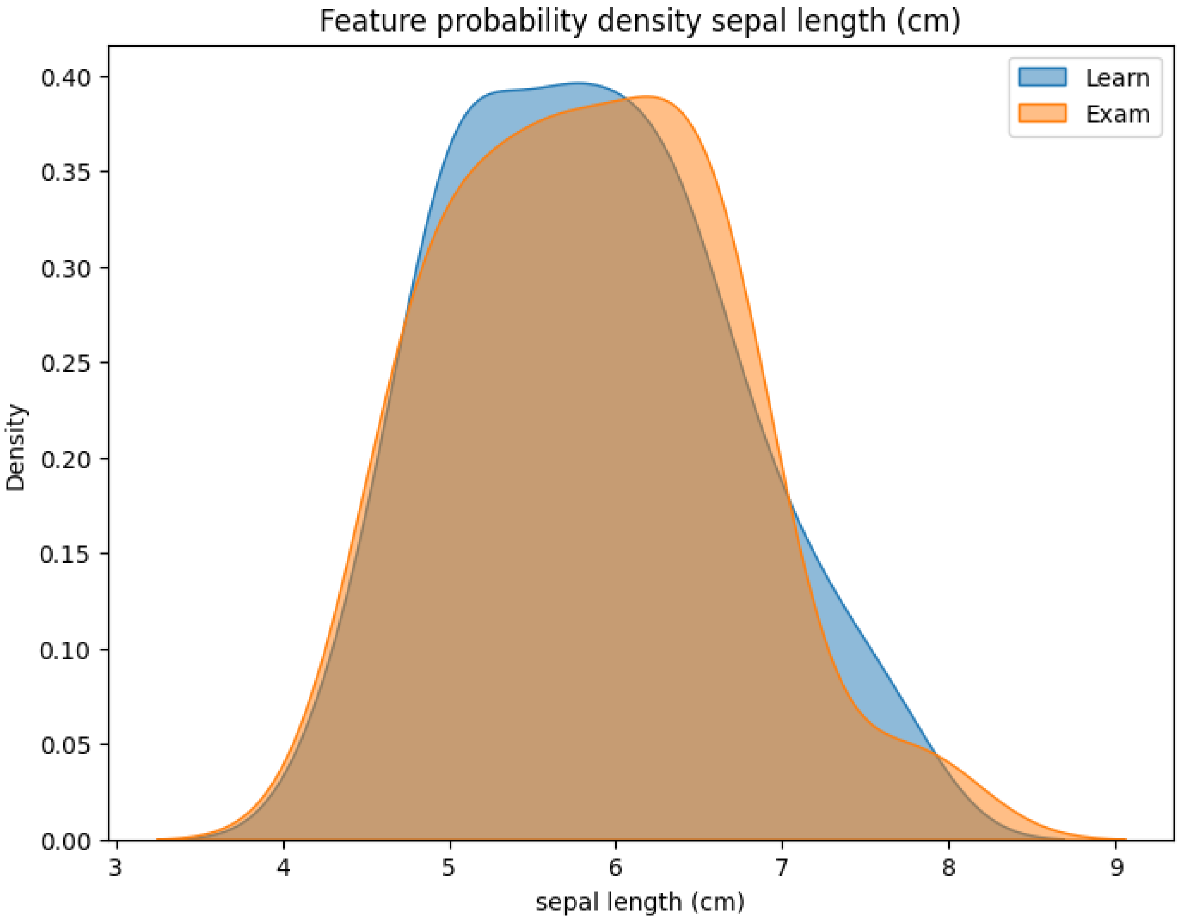
```python
        subset[iris.feature_names[1]], subset[iris.feature_names[2]], label=species)

# Label axes
ax.set_xlabel(iris.feature_names[0])
ax.set_ylabel(iris.feature_names[1])
ax.set_zlabel(iris.feature_names[2])

# Title and legend
ax.set_title("Three-dimensional data dispersion")
ax.legend()
plt.show()

# Heatmap of feature correlations
plt.figure(figsize=(10, 8))
sns.heatmap(data_combined[iris.feature_names].corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Feature correlation heat map")
plt.show()

# Kernel Density Estimation (KDE) plot for each feature
for feature in iris.feature_names:
    plt.figure(figsize=(8, 6))
    sns.kdeplot(data_train[feature], label='Train', fill=True, alpha=0.5)  # Train data distribution
    sns.kdeplot(data_test[feature], label='Test', fill=True, alpha=0.5)  # Test data distribution
    plt.xlabel(feature)
    plt.title(f"Feature probability density {feature}")  # Title indicating which feature is displayed
    plt.legend()
    plt.show()

# Discretization of the first feature (sepal length) into categorical bins
bins = [data_combined[iris.feature_names[0]].min(), 5.5, 6.5, data_combined[iris.feature_names[0]].max()]  # Define bin edges
labels = ['short', 'medium', 'tall']  # Labels for each category

# Create a new categorical column based on the feature value
data_combined['discrete_feature'] = pd.cut(data_combined[iris.feature_names[0]], bins=bins, labels=labels)

# Display first five rows with the new categorical feature
print(data_combined[['discrete_feature', iris.feature_names[0]]].head())

# Extract only the data for the "Setosa" species (target = 0)
setosa_data = data_combined[data_combined['target'] == 0]

# Display summary statistics for the Setosa subset
print(setosa_data.describe())
```

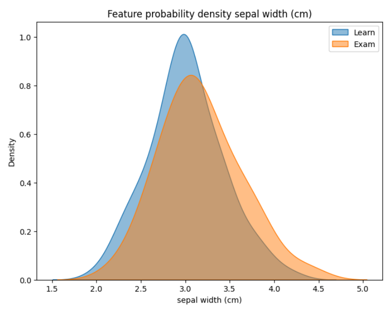
ممنون از توجه شما.


Scatter of samples based on two characteristics
setosa
versicolor
virginica
sepal width (cm)
sepal length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

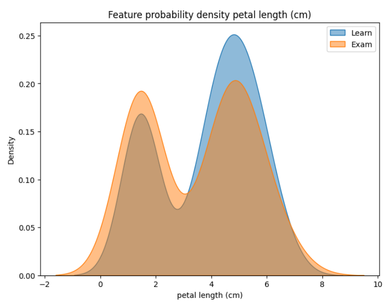

Three-dimensional data dispersion
setosa
versicolor
virginica
sepal length (cm)
sepal width (cm)
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
2.0
2.5
3.0
3.5
4.0
4.5
1
2
3
4
5
6
7


Feature correlation heat map
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
1
-0.12
0.87
0.82
-0.12
1
-0.43
-0.37
0.87
-0.43
1
0.96
0.82
-0.37
0.96
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

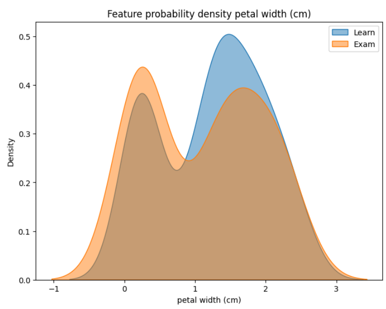




(آ) تصویر ۱



(ب) تصویر ۲



(ج) تصویر ۳