
**تمرین یادگیری ماشین دوره کارشناسی ارشد**

**رشته مهندسی مکاترونیک**

عنوان

# تمرین درس یادگیری ماشین

نگارش

**علیرضا امیری**

اسفند ۱۴۰۳

# فصل ۱

# پاسخ سوالات سری اول

Folder Drive Google

## ۱.۱ پاسخ سوال ۱

### ۱.۱.۱ قسمت اول، بخش الف

برای محاسبه ابعاد ماتریس ها پس از ضرب دو ماتریس در یک دیگر، در صورتی که ابعاد ماتریس اول X برابر با $m * n$ و ابعاد ماتریس دوم Y برابر $u * v$ باشد، می توانیم از رابطه ی زیر استفاده می کنیم.

$$X_{m \times n} \cdot Y_{u \times v} = Z_{m \times v}$$

برای ماتریس های داده شده داریم:

$$size(A) = ۲ * ۳, size(B) = ۴ * ۲$$

در نتیجه، سایز ماتریس های داده شده برابر خواهد بود با:

$$size(B \times A) = ۴ * ۳$$

$$size(B^T) = ۲ * ۴$$

$$size(B^T \times A) = Null$$

$$size(A^T \times B) = Null$$

ماتریس هایی که قابل ضرب شدن نیستند، به دلیل تناقض در ابعاد ماتریس ها و عدم همخوانی تعداد ستون های ماتریس اول با تعداد سطر های ماتریس دوم می باشد.

### ۲.۱.۱ قسمت اول، بخش ب

ماتریس های مطرح شده در بخش قبل به صورت زیر خواهند بود.

$$BA = \begin{bmatrix} b_{۱۱} & b_{۱۲} \\ b_{۲۱} & b_{۲۲} \\ b_{۳۱} & b_{۳۲} \\ b_{۴۱} & b_{۴۲} \end{bmatrix} \begin{bmatrix} a_{۱۱} & a_{۱۲} & a_{۱۳} \\ a_{۲۱} & a_{۲۲} & a_{۲۳} \end{bmatrix} = \begin{bmatrix} b_{۱۱}a_{۱۱} + b_{۱۲}a_{۲۱} & b_{۱۱}a_{۱۲} + b_{۱۲}a_{۲۲} & b_{۱۱}a_{۱۳} + b_{۱۲}a_{۲۳} \\ b_{۲۱}a_{۱۱} + b_{۲۲}a_{۲۱} & b_{۲۱}a_{۱۲} + b_{۲۲}a_{۲۲} & b_{۲۱}a_{۱۳} + b_{۲۲}a_{۲۳} \\ b_{۳۱}a_{۱۱} + b_{۳۲}a_{۲۱} & b_{۳۱}a_{۱۲} + b_{۳۲}a_{۲۲} & b_{۳۱}a_{۱۳} + b_{۳۲}a_{۲۳} \\ b_{۴۱}a_{۱۱} + b_{۴۲}a_{۲۱} & b_{۴۱}a_{۱۲} + b_{۴۲}a_{۲۲} & b_{۴۱}a_{۱۳} + b_{۴۲}a_{۲۳} \end{bmatrix}$$

$$B^T = \begin{bmatrix} b_{۱۱} & b_{۲۱} & b_{۳۱} & b_{۴۱} \\ b_{۱۲} & b_{۲۲} & b_{۳۲} & b_{۴۲} \end{bmatrix}$$

### ۳.۱.۱ قسمت دوم، بخش اول

با در نظر داشتن ابعاد ماتریس x برابر با ۲ * ۱ و ابعاد ماتریس $\theta$ برابر با ۱ * ۲، ابعاد ماتریس $x * \theta$ برابر با ۱ * ۱ خواهد بود. با افزایش تعداد نمونه ها در سطر های جدیدی برای ماتریس $\mathbf{X}$، ابعاد ماتریس $\mathbf{X} \times \theta$ برابر

$۱ * n$ خواهد بود.

### ۴.۱.۱ قسمت دوم، بخش دوم

$$X\theta - \vec{y} = \begin{bmatrix} x^{(۱)T}\theta \\ \vdots \\ x^{(n)T}\theta \end{bmatrix} - \begin{bmatrix} y^{(۱)} \\ \vdots \\ y^{(n)} \end{bmatrix} = \begin{bmatrix} h_\theta(x^{(۱)}) - y^{(۱)} \\ \vdots \\ h_\theta(x^{(n)}) - y^{(n)} \end{bmatrix}.$$

آنگاه با در نظر داشتن آنکه $z$، خواهیم داشت $z^T z = \sum_i z_i^۲$:

$$\frac{۱}{۲}(X\theta - \vec{y})^T(X\theta - \vec{y}) = \frac{۱}{۲}\sum_{i=۱}^{n}(h_\theta(x^{(i)}) - y^{(i)})^۲ = J(\theta)$$

سپس، برای کاهش مقدار $J$ مشتق آن را نسبت به $\theta$ به دست می آوریم.

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{۱}{۲}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

$$= \frac{۱}{۲}\nabla_\theta \left((X\theta)^T X\theta - (X\theta)^T\vec{y} - \vec{y}^T(X\theta) + \vec{y}^T\vec{y}\right)$$

$$= \frac{۱}{۲}\nabla_\theta \left(\theta^T(X^T X)\theta - \vec{y}^T(X\theta) - \vec{y}^T(X\theta)\right)$$

$$= \frac{۱}{۲}\nabla_\theta \left(\theta^T(X^T X)\theta - ۲(X^T\vec{y})^T\theta\right)$$

$$= \frac{۱}{۲}\left(۲X^T X\theta - ۲X^T\vec{y}\right)$$

$$= X^T X\theta - X^T \vec{y}$$

## ۲.۱ پاسخ سوال ۲

### ۱.۲.۱ پرده اول

با در اختیار داشتن فرمول احتمال بیز به صورت زیر خواهیم داشت:

$$P(+) = P(+ \cap \text{sick}) + P(+ \cap notsick)$$

در این رابطه، احتمال کل مثبت بودن نتیجه آزمایش در حالت مریض بودن یا نبودن محاسبه می شود.

$$P(+) = P(+|\text{sick})P(\text{sick}) + P(+|notsick)P(notsick)$$

با جایگذاری مقادیر عددی خواهیم داشت:

$$P(+) = \frac{۹۹}{۱۰۰} \cdot ۱۰^{-۴} + \frac{۱}{۱۰۰} \cdot \frac{۹۹۹۹}{۱۰۰۰۰} = ۰/۰۰۱۰۰۹۸$$

حال با استفاده از فرمول بیز خواهیم داشت:

$$P(\text{sick}|+) = \frac{P(+|\text{sick})P(\text{sick})}{P(+)}$$

$$P(\text{sick}|+) = \frac{۰/۰۰۰۰۹۹}{۰/۰۰۱۰۰۹۸} \approx ۰/۰۹۸۰۳۹۲$$

### ۲.۲.۱ پرده دوم

در این پرده، به محاسبه ی احتمال بیمار بودن علی در صورتی که پاسخ هر دو تست مثبت باشند خواهیم پرداخت.

$$P(\text{sick}|++) =?$$

مجددا با استفاده از قانون احتمال کل خواهیم داشت:

$$P(++) = P(++|\text{sick})P(\text{sick}) + P(++|notsick)P(notsick)$$

که با جایگذاری مقادیر احتمال ها خواهیم داشت:

$$P(++) = \frac{۹۹}{۱۰۰} \cdot \frac{۹۹۹۹}{۱۰۰۰۰} \cdot ۱۰^{-۴} + \frac{۱}{۱۰۰} \cdot \frac{۱}{۱۰۰۰۰} \cdot \frac{۹۹۹۹}{۱۰۰۰۰}$$

$$= ۰/۰۰۰۰۹۸۹۹ + ۰/۰۰۰۰۰۰۰۰۹۹۹$$

$$= ۰/۰۰۰۰۹۹۹۹۹ \approx ۰/۰۰۰۱$$

در پایان، با استفاده از رابطه ی بیز چنین به دست می آوریم:

$$P(\text{sick}|++) = \frac{P(++|\text{sick})P(\text{sick})}{P(++)}$$

در اینجا خواهیم داشت:

$$P(\text{sick}|++) = \frac{P(++|\text{sick})P(\text{sick})}{P(++)}$$

و با جایگذاری مقادیر به دست آمده در رابطه ی بالا خواهیم داشت:

$$= \frac{۰/۹۸۹۹۰۱ \times ۱۰^{-۴}}{۰/۰۰۰۱}$$

$$= ۰/۹۸۹۹$$

$$P(\text{sick}|++) \approx ۰/۹۸۹۹$$

### ۳.۲.۱ پرده سوم

$$P(\text{sick}|++-) = \frac{P(++-|\text{sick})P(\text{sick})}{P(++-)}$$

ابتدا به محاسبه ی $P(++-)$ می پردازیم:

$$P(++-) = P(++-|\text{sick})P(\text{sick}) + P(++-|notsick)P(notsick)$$

با جایگذاری مقادیر به دست می آوریم:

$$= (۰/۹۹ \cdot ۰/۹۹۹۹ \cdot ۰/۰۰۰۰۰۱) \cdot ۱۰^{-۴} + (۰/۰۱ \cdot ۰/۰۰۰۱ \cdot \frac{۹۹۹۹۹۹}{۱۰۰۰۰۰۰}) \cdot \frac{۹۹۹۹}{۱۰۰۰۰}$$

$$= ۰/۰۰۰۰۰۰۹۸۹۹۰۱ \cdot ۰/۰۰۰۱ + ۰/۰۰۰۰۰۰۹۹۹ \cdot ۰/۹۹۹۹$$

$$\approx ۰/۰۰۰۰۰۰۹۹۸$$

$$P(\text{sick}|++-) = \frac{۰/۰۰۰۰۰۰۹۸۹۹۰۱ \cdot ۱۰^{-۴}}{۰/۰۰۰۰۰۰۹۹۸}$$

$$\approx ۹/۹۱۸۸۴۷ \cdot ۱۰^{-۵}$$

# ۳.۱ پاسخ سوال ۳، قسمت اول

## ۱.۳.۱ بخش الف، دریافت داده

### ۱ ۱.۱.۳.۱

برای این قسمت، فایل ۱۴۹ از مجموعه داده های قرار داده شده در سایت توسط نرم افزار های مدیریت دانلود، دانلود می شود. فرمت این فایل $mat$. است. برای استفاده از این فایل، ابتدا آن را در فضای ابری درایو ذخیره کرده و با ایجاد دسترسی برای آن فایل و استفاده از دستور ،gdown آن را در محیط گوگل کولب وارد می کنیم. در ادامه، با استفاده از دستور loadmat از پکیج ،scipy.io می توانیم دیتا را در یک متغیر ذخیره می کنیم.

```
!pip install --upgrade --no-cache-dir gdown
```

```
!gdown 1zKj4N5nFFKK1u9AhzlqJLvsw-xRC0kz_
```

```
from scipy.io import loadmat
file_path = '/content/149.mat'
mat_data = loadmat(file_path)
```

### ۲ ۲.۱.۳.۱

پس از ذخیره سازی فایل، در متغیر بالا، می توانیم به جستجو درباره ویژگی های آن بپردازیم. با استفاده از دستور type مشاهده می کنیم که متغیر ذخیره شده از جنس dictionary است. یک دیکشنری متشکل از بخش های زیر است:

۱. **Key** کلید، مشخصه ای منحصر به فرد به ازای هر مقدار در دیکشنری است.

۲. **value** حاوی مقدار عددی مرتبط با یک key است و می تواند مقادیر تکراری داشته باشد.

۳. **item** به زوج key و value متناظر آن یک item گفته می شود و با علامت : با یکدیگر مرتبط می شوند.

۳ ۳.۱.۳.۱

در این بخش، با مشاهده ی کلید های موجود در دیکشنری سیستم خواهیم دید که این مجموعه داده توسط کلید های زیر تعریف شده اند. ،__header__ ،__version__ ،__globals__ ،۱۴۹X _DE_time،۱۴۹X _FE_time، RPM۱۴۹X و از میان این مقادیر، دو سیگنال $X۱۴۹\_DE\_time$ و $X۱۴۹\_FE\_time$ حاوی سیگنال های اطلاعات هستند. در این بخش سیگنال $X۱۴۹\_DE\_time$ انتخاب و در متغیری ذخیره می شود.

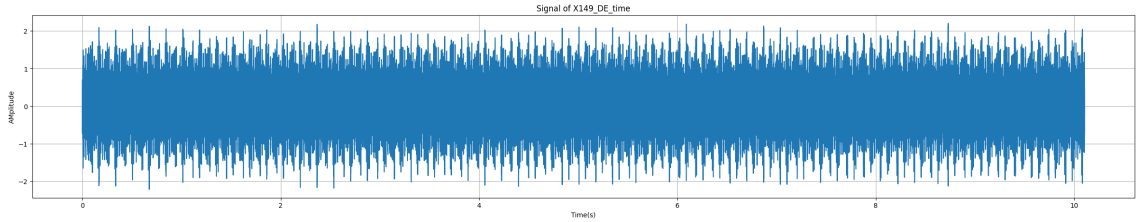
## ۲.۳.۱ بخش ب، نمایش سیگنال

برای نمایش سیگنال انتخاب شده، لازم از کتابخانه ی Matplotlib استفاده شود. برای نمایش سیگنال، علاوه بر در اختیار داشتن خود سیگنال باید مقدار محور افقی که در این سوال برابر با زمان خواسته شده است نیز محاسبه شود. با در نظر داشتن فرکانس نمونه برداری برابر با ۴۸۰۰۰۰، مقادیر محور افقی با استفاده از دستور linspace در پکیج numpy ایجاد می شود. با تنظیم ابعاد نمودار مورد نظر با استفاده از دستور figure و تعیین ابعاد نمودار، می توانیم نمودار مورد نظر را نمایش دهیم.

```
import numpy as np
import matplotlib.pyplot as plt

fs = 48000 #KHz
time = np.linspace(0, len(DE_time)/fs , len(DE_time))

plt.figure(figsize=(30,5))
plt.plot(time, DE_time)
plt.xlab
("Time(s)")
plt.ylabel("AMplitude")
plt.title("Signal of X149_DE_time")
```

```
plt.grid(True)
plt.show()
```

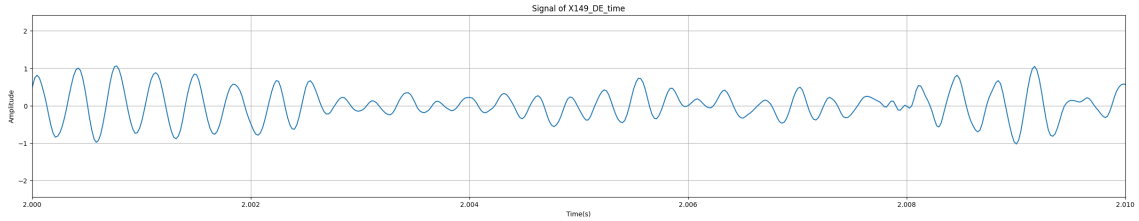
با اجرای این کد، نمودار سیگنال به صورت زیر نمایش داده می شود.



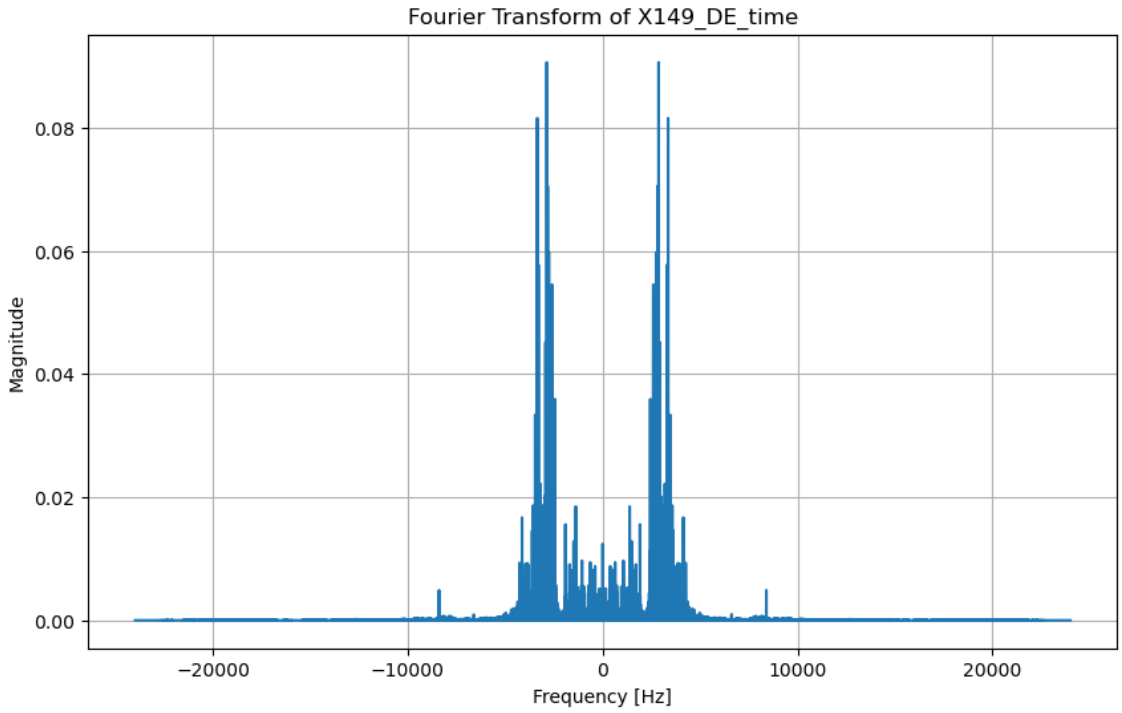

شکل ۱.۱: نمودار سیگنال $X۱۴۹_{D}E_{t}ime$

در بخش بعد، با محدود کردن ناحیه ی x در نمودار بین بازه ی ۲ تا ۰۱.۲ ثانیه، سیگنال را مشاهده می کنیم.

```
time = np.linspace(0, len(DE_time)/fs, len(DE_time))
plt.figure(figsize=(30, 5))
plt.plot(time, DE_time)
plt.xlabel("Time(s)")
plt.xlim(2, (01.2
plt.ylabel("Amplitude")
plt.title("Signal of X149_DE_time")
plt.grid(True)
plt.show()
```




شکل ۲.۱: نمودار در بازه ی۲ تا ۲٫۰۱ ثانیه




شکل ۳.۱: نمودار حوزه فرکانس سیگنال مورد بررسی

### ۳.۳.۱ بخش ج، تحلیل فرکانسی

در این بخش، تابعی برای محاسبه ی تبدیل فوریه نوشته می شود. در نوشتن این تابع از روش fft در پکیج numpy استفاده شده است و پس از مشخص کردن فرکانس نمونه برداری داده ها، نمودار حوزه فرکانس سیستم رسم شده است.

همچنین، با اندازه گیری فرکانس با بیشترین دامنه، فرکانس غالب سیستم را برابر با $۳۲۰۱Hz$ به دست می آوریم.

### ۴.۳.۱ بخش د و ه، تقسیم بندی سیگنال

برای تقسیم سیگنال به سطر هایی با ۱۲۸ نمونه، از دستور reshape استفاده می کنیم. با این حال، باید توجه داشته باشیم که تعداد داده ها باید حتما بر ۱۲۸ بخش پذیر باشند و بنابراین، پیش از اعمال دستور reshape، سیگنال را به تعدادی بخش پذیر بر ۱۲۸ برش می دهیم. در نهایت، ماتریسی به صورت زیر به دست می آید.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.043809 | -0.076770 | -0.046938 | 0.047773 | 0.167727 | 0.274955 | 0.367580 | 0.422029 | 0.435798 | 0.377177 | ... | -0.389902 | -0.170022 | 0.064045 | 0.267862 | 0.423698 | 0.480024 | 0.425993 | 0.279962 | 0.065505 | -0.157296 |
| 1 | -0.325023 | -0.423072 | -0.433086 | -0.368206 | -0.252007 | -0.097423 | 0.046521 | 0.157922 | 0.194847 | 0.154375 | ... | 0.342964 | 0.417857 | 0.427453 | 0.383435 | 0.298320 | 0.165641 | 0.003129 | -0.159591 | -0.286638 | -0.360696 |
| 2 | -0.424324 | -0.480441 | -0.534890 | -0.527171 | -0.422446 | -0.267236 | -0.077396 | 0.080317 | 0.207364 | 0.347136 | ... | -1.197035 | -0.995304 | -0.658390 | -0.236987 | 0.196098 | 0.579534 | 0.836548 | 0.943567 | 0.898924 | 0.714925 |
| 3 | 0.437049 | 0.122457 | -0.162929 | -0.376968 | -0.496087 | -0.505684 | -0.408052 | -0.230937 | -0.031918 | 0.119745 | ... | -0.467924 | -0.577030 | -0.568060 | -0.467716 | -0.310002 | -0.172525 | -0.074893 | -0.007510 | 0.033587 | 0.063210 |
| 4 | 0.070512 | 0.050902 | 0.021279 | -0.001252 | 0.012726 | 0.079065 | 0.162094 | 0.224262 | 0.239073 | 0.226139 | ... | 0.288098 | 0.331490 | 0.328361 | 0.275790 | 0.173151 | 0.057369 | -0.031084 | -0.089287 | -0.114947 | -0.112235 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3784 | 0.027329 | 0.048399 | 0.064254 | 0.100344 | 0.133931 | 0.135183 | 0.112861 | 0.083655 | 0.078231 | 0.094503 | ... | -0.208198 | -0.258892 | -0.338791 | -0.408678 | -0.413684 | -0.335871 | -0.167310 | 0.021905 | 0.179409 | 0.304787 |
| 3785 | 0.435798 | 0.557003 | 0.607697 | 0.528006 | 0.316470 | 0.037968 | -0.245540 | -0.447689 | -0.566599 | -0.605610 | ... | 0.584749 | 0.376968 | 0.177532 | 0.009805 | -0.096380 | -0.158130 | -0.152289 | -0.091791 | -0.002086 | 0.116407 |
| 3786 | 0.207364 | 0.233232 | 0.182747 | 0.094294 | 0.015646 | -0.039428 | -0.071555 | -0.064462 | 0.003964 | 0.126212 | ... | 0.031710 | 0.059664 | 0.083446 | 0.114113 | 0.144988 | 0.209658 | 0.300406 | 0.382601 | 0.439344 | 0.444977 |
| 3787 | 0.419943 | 0.363408 | 0.273286 | 0.138938 | -0.038802 | -0.232398 | -0.399498 | -0.490872 | -0.485239 | -0.374882 | ... | 0.050694 | -0.135183 | -0.308542 | -0.419943 | -0.436632 | -0.334410 | -0.135183 | 0.112652 | 0.351726 | 0.530092 |
| 3788 | 0.640658 | 0.650046 | 0.556169 | 0.390319 | 0.176071 | -0.022113 | -0.189423 | -0.317513 | -0.380514 | -0.396578 | ... | 0.922497 | 0.470845 | -0.053406 | -0.565974 | -0.923332 | -1.121725 | -1.136537 | -0.984039 | -0.694689 | -0.274121 |

3789 rows × 128 columns

شکل ۴.۱: دیتافریم بخش های سیستم

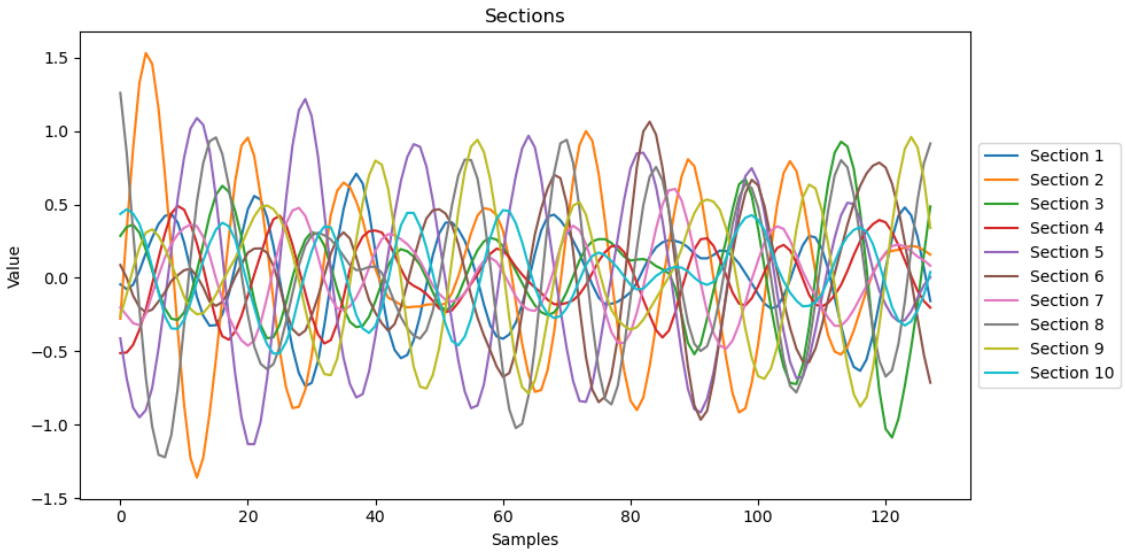
```
num_sections = len(DE_time) // 128
signal_trimmed = DE_time[:num_sections * 128]
sections = signal_trimmed.reshape(1-,128)
df = pd.DataFrame(sections)
df
```

در نهایت، برای رسم نمودار بخش های انتخاب شده از سیگنال، با استفاده از کتابخانه ی matplotlib، آنها را در یک حلقه ی for به نمودار اضافه کرده و در نهایت نمودار را نمایش می دهیم. لازم به ذمر است که برای اضافه کردن legend به سیستم نیز، برای هر نمودار label متناسب تولید کرده و در نهایت، آن را در legend نمایش می دهیم.

```
plt.figure(figsize=(10, 5))
for i in range(10):
var = df.iloc[13*i]
var = np.array(var).reshape(var.shape[0], (1-
plt.plot(np.arange(len(var)), var.flatten(), label=f"Section {i+1}")
plt.title("Sections")
plt.xlabel("Samples")
plt.ylabel("Value")
plt.legend(loc="center left", bbox_to_anchor=(1, ((5.0
plt.tight_layout()
```

```
plt.show()
```

نمودار به دست آمده آنگاه به صورت زیر خواهد بود.




شکل ۵.۱: نمودار بخش های جدا شده از سیستم بر حسب زمان

### ۵.۳.۱ بخش و، استخراج ویژگی

در این قسمت، ویژگی های میانگین، انحراف معیار و RMS را با استفاده از روش های موجود در کتابخانه ی numpy محاسبه می کنیم. برای این کار، در یک class، تابع های مورد نیاز را تعریف کرده و سپس در پایان آن، در تابع main، هر سه را فراخوانی می کنیم. برای محاسبه ی میانگین و انحراف معیار از روش های آماده در کتابخانه ی numpy استفاده شده و RMS به صورت مجزا تعریف شده است.

```
class Statistics:
def __init__(self, signal):
self.signal = np.array(signal)


def mymean(self):
return np.mean(self.signal)
```

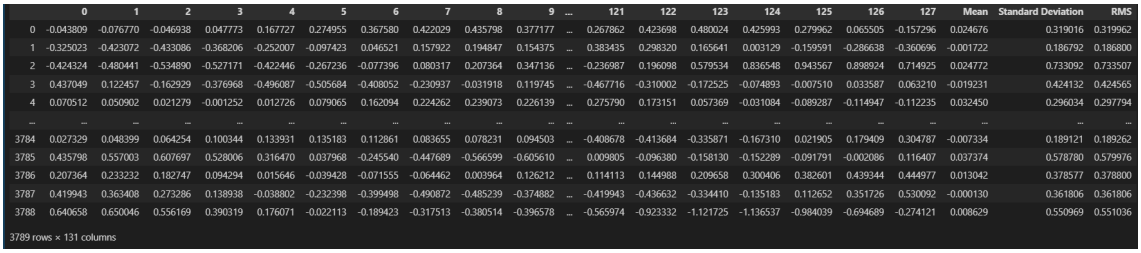
```
def mystddev(self):
return np.std(self.signal)

def myRMS(self):
return np.sqrt(np.mean(np.square(self.signal)))

def main(signal):
stats = Statistics(signal)
print("Mean:", stats.mymean())
print("Standard Deviation:", stats.mystddev())
print("RMS:", stats.myRMS())
```

آنگاه با فراخوانی این تابع و اعمال ورودی سیگنال به آن، می توانیم جدول ویژکی ها را محاسبه کنیم.

برای ایجاد ویژگی های خواسته شده برای تمام نمونه ها در دیتافریم، با فرض اینکه آن ستون ها وجود دارند، مقادیر را مطابق با روش های موجود در کلاس ساخته شده و به وسیله ی $lamda function$ می نویسیم.

```
    df['Mean'] = df.apply(lambda row: Statistics(row.values).mymean(), axis = 1)

df['Standard Deviation'] = df.apply(lambda row: Statistics(row.values).mystddev(), ax
df['RMS'] = df.apply(lambda row: Statistics(row.values).myRMS(), axis = 1)
df
df.tocsv("dataset.csv", index=False)
```

در پایان، دیتافریم به صورت زیر قابل مشاهده خواهد بود.



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 121 | 122 | 123 | 124 | 125 | 126 | 127 | Mean | Standard Deviation | RMS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.043809 | -0.076770 | -0.046938 | 0.047773 | 0.167727 | 0.274955 | 0.367580 | 0.422029 | 0.435798 | 0.377177 | ... | 0.267862 | 0.423698 | 0.480024 | 0.425993 | 0.279962 | 0.065505 | -0.157296 | 0.024676 | 0.319016 | 0.319962 |
| 1 | -0.325023 | -0.423072 | -0.433086 | -0.368206 | -0.252007 | -0.097423 | 0.046521 | 0.157922 | 0.194847 | 0.154375 | ... | 0.383435 | 0.298320 | 0.165641 | 0.003129 | -0.159591 | -0.286638 | -0.360696 | -0.001722 | 0.186792 | 0.186800 |
| 2 | -0.424324 | -0.480441 | -0.534890 | -0.527171 | -0.422446 | -0.267236 | -0.077396 | 0.080317 | 0.207364 | 0.347136 | ... | -0.236987 | 0.196098 | 0.579534 | 0.836548 | 0.943567 | 0.898924 | 0.714925 | 0.024772 | 0.733092 | 0.733507 |
| 3 | 0.437049 | 0.122457 | -0.162929 | -0.376968 | -0.496087 | -0.505684 | -0.408052 | -0.230937 | -0.031918 | 0.119745 | ... | -0.467716 | -0.310002 | -0.172525 | -0.074893 | -0.007510 | 0.033587 | 0.063210 | -0.019231 | 0.424132 | 0.424565 |
| 4 | 0.070512 | 0.050902 | 0.021279 | -0.001252 | 0.012726 | 0.079065 | 0.162094 | 0.224262 | 0.239073 | 0.226139 | ... | 0.275790 | 0.173151 | 0.057369 | -0.031084 | -0.089287 | -0.114947 | -0.112235 | 0.032450 | 0.296034 | 0.297794 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3784 | 0.027329 | 0.048399 | 0.064254 | 0.100344 | 0.133931 | 0.135183 | 0.112861 | 0.083655 | 0.078231 | 0.094503 | ... | -0.408678 | -0.413684 | -0.335871 | -0.167310 | 0.021905 | 0.179409 | 0.304787 | -0.007334 | 0.189121 | 0.189262 |
| 3785 | 0.435798 | 0.557003 | 0.607697 | 0.528006 | 0.316470 | 0.037968 | -0.245540 | -0.447689 | -0.566599 | -0.605610 | ... | 0.009805 | -0.096380 | -0.158130 | -0.152289 | -0.091791 | -0.002086 | 0.116407 | 0.037374 | 0.578780 | 0.579976 |
| 3786 | 0.207364 | 0.233232 | 0.182747 | 0.094294 | 0.015646 | -0.039428 | -0.071555 | -0.064462 | 0.003964 | 0.126212 | ... | 0.114113 | 0.144988 | 0.209658 | 0.300406 | 0.382601 | 0.439344 | 0.444977 | 0.013042 | 0.378577 | 0.378800 |
| 3787 | 0.419943 | 0.363408 | 0.273286 | 0.138938 | -0.038802 | -0.232398 | -0.399498 | -0.490872 | -0.485239 | -0.374882 | ... | -0.419943 | -0.436632 | -0.334410 | -0.135183 | 0.112652 | 0.351726 | 0.530092 | -0.000130 | 0.361806 | 0.361806 |
| 3788 | 0.640658 | 0.650046 | 0.556169 | 0.390319 | 0.176071 | -0.022113 | -0.189423 | -0.317513 | -0.380514 | -0.396578 | ... | -0.565974 | -0.923332 | -1.121725 | -1.136537 | -0.984039 | -0.694689 | -0.274121 | 0.008629 | 0.550969 | 0.551036 |

3789 rows × 131 columns

شکل ۶.۱: دیتافریم پس از استخراج ویژگی

در نهایت، دیتافریم را در یک فایل csv ذخیره می کنیم.

# ۴.۱ پاسخ سوال سوم، قسمت دوم

## ۱.۴.۱ بخش اول، بررسی اولیه

دیتاست گل زنبق اولین بار توسط رونالد فیشر در کتابش معرفی شد و شامل داده های ۱۵۰ نمونه گل زنبق است. ویژگی های اندازه گیری شده برای این گل ها به شرح زیر است:

۱. Length Sepal

۲. Width Sepal

۳. Length Petal

۴. Width Petal

که مربوط به ابعاد اجزای گل ها می باشد. علاوه بر این، سه کلاس مختلف این گل ها به صورت زیر در این دیتاست نام گذاری شده اند:

۱. Setosa :۰

۲. Versicolor :۱

۳. Virginica :۲

از ویژگی های بارز این دیتاست آن است که به ازای هر کلاس، دقیقا ۵۰ نمونه داده وجود دارد که دیتاست را متعادل می سازد. علاوه بر این، داده ی خالی ندارد و همچنین کلاس Setosa به صورت خطی قابل جداسازی است. برای استفاده از این دیتاست، می توانیم آن را از کتابخانه ی scikit-learn فراخوانی کنیم. در این دستور، از مجموعه داده های موجود در کتابخانه ی $scikit-learn$ دیتاست Iris را فراخوانده و سپس آن را در یک دیتافریم ذخیره می کنیم.

```
from sklearn import datasets
import pandas as pd

# Load dataset
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target

print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

شکل ۷.۱: نمونه داده ی دیتاست زنبق

در ادامه برای تقسیم دیتاست به بخش آموزشی و تست، با استفاده از روش $Train_Test_split$ از کتابخانه ی sklearn در بخش $model_selection$ داده ها را به دو بخش تقسیم می کنیم. در اینجا ابتدا مقادیر فیچرها و کلاس های خروجی را به عنوان ورودی و خروجی سیستم تعریف کرده و سپس با استفاده از دستور ذکر شده، آنها را تقسیم می کنیم.

```
        from sklearn.model_selection import train_test_split

        X = iris.data
        y = iris.target

        X_train, X_test, y_train, y_test =
        train_test_split(X,y, test_size=0.15, random_state=43)
```

در ادامه، نام ستون های متغیر های به دست آمده را باید مجدد نام گذاری کنیم و البته آنها را در دیتافریم هایی ذخیره کنیم.

```
X_train = pd.DataFrame(X_train)
X_train.rename(columns={0:'Sepal Length',
 1:"Sepal Width",
 2:"Petal Length",
 3:"Petal Width"},inplace=True)

X_test = pd.DataFrame(X_test)
X_test.rename(columns={0:'Sepal Length',
 1:"Sepal Width",
 2:"Petal Length",
 3:"Petal Width"},inplace=True)

y_train = pd.DataFrame(y_train)
y_train.rename(columns={0:'Iris Type'},inplace=True)

y_test = pd.DataFrame(y_test)
```

```
y_test.rename(columns={0:'Iris Type'},inplace=True)
```

در نهایت، دیتافریم های آموزش و تست را با به هم پیوستن داده های x و y آنها به وسیله دستور concat ترکیب می کنیم.

```
Train_df = pd.concat([X_train,y_train], axis=1)
Test_df = pd.concat([X_test,y_test], axis=1)
```

برای مشخص کردن آنکه داده ها، مربوط به تست و یا آموزش هستند، ستونی به هر یک از دیتافریم ها با عنوان $istrain$ اضافه شده که برای داده های train دارای مقدار ۱ و برای داده های تست دارای مقدار ۰ است. در نهایت، با استفاده ی مجدد از دستور concat و به هم پیوستن این دو دیتافریم، دیتاست به صورت زیر به دست می آید.

```
temporary_train = Train_df
temporary_train["is train"] = 1
temporary_test = Test_df
temporary_test["is train"] = 0

Dataset = pd.concat([temporary_train,temporary_test],ignore_index=True)
```

| | Sepal Length | Sepal Width | Petal Length | Petal Width | Iris Type | is train |
|---|---|---|---|---|---|---|
| 0 | 6.2 | 2.9 | 4.3 | 1.3 | 1 | 1 |
| 1 | 5.7 | 4.4 | 1.5 | 0.4 | 0 | 1 |
| 2 | 4.8 | 3.0 | 1.4 | 0.1 | 0 | 1 |
| 3 | 5.8 | 2.8 | 5.1 | 2.4 | 2 | 1 |
| 4 | 6.3 | 3.3 | 6.0 | 2.5 | 2 | 1 |
| ... | ... | ... | ... | ... | ... | ... |
| 145 | 4.4 | 3.0 | 1.3 | 0.2 | 0 | 0 |
| 146 | 6.8 | 2.8 | 4.8 | 1.4 | 1 | 0 |
| 147 | 7.7 | 2.8 | 6.7 | 2.0 | 2 | 0 |
| 148 | 4.8 | 3.4 | 1.6 | 0.2 | 0 | 0 |
| 149 | 6.0 | 2.9 | 4.5 | 1.5 | 1 | 0 |

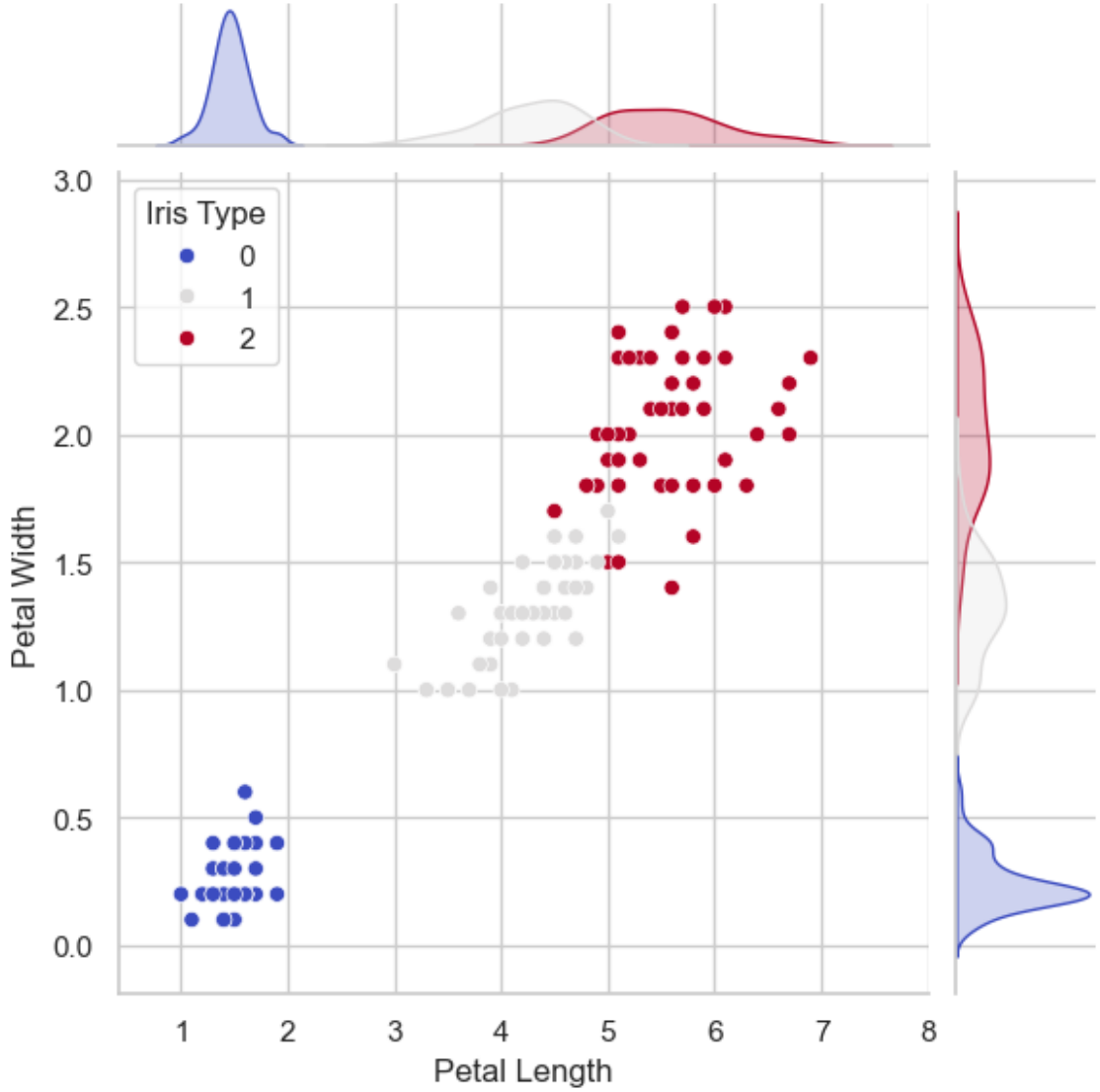
150 rows × 6 columns

شکل ۸.۱: دیتاست

### ۲.۴.۱ تحلیل بصری داده

در این بخش با در اختیار داشتن دیتاست، به رسم نمودار هایی از دیتاست و مشاهده داده های موجود در آن می پردازیم. در بخش اول، دو ویژگی $PetalLength$ و $PetalWidth$ از دیتاست را انتخاب کرده و توزیع آنها را نمایش می دهیم. این کار با استفاده از دستور jointplot از کتابخانه ی seaborn انجام می شود.

```
import seaborn as sns
sns.jointplot(x=Dataset["Petal Length"], y=Dataset["Petal Width"],

kind="scatter", hue=Dataset["Iris Type"],palette="coolwarm")
```

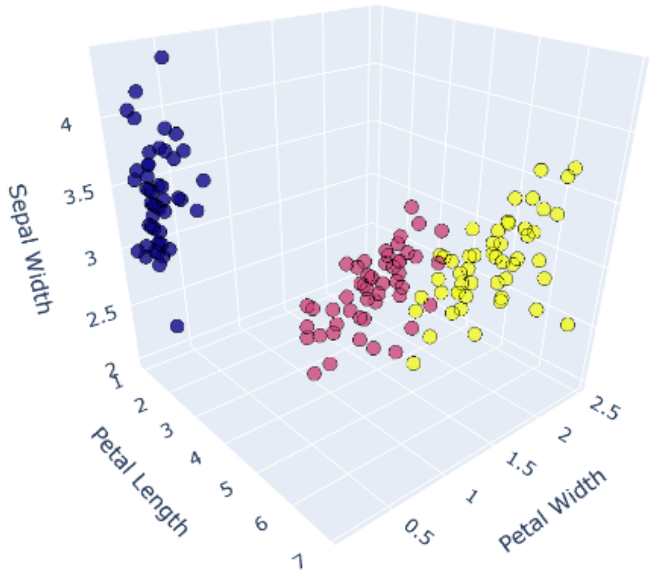



شکل ۹.۱: توزیع داده های $PetalLength$ و $PetalWidth$

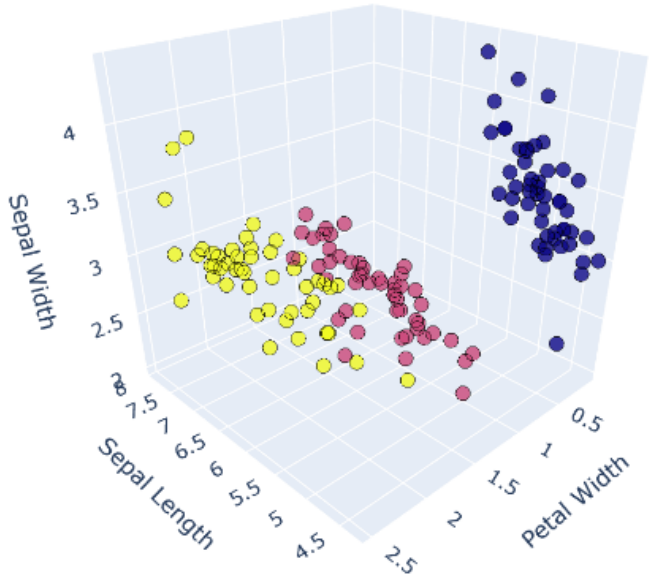
در ادامه برای نمایش سه ویژگی از این دیتاست در یک نمودار سه بعدی به تفکیک نوع زنبق ها، از کتابخانه ی plotly استفاده می کنیم. در این روش، مقادیر دیتافریم در سه متغیر x و y و z تعریف شده و رنگ آن بر اساس نوع زنبق تعیین می شود. پس از تعیین نمودار و برای بهبود آن، مارکر های قرار داده شده را تنظیم می کنیم و رنگ حاشیه آنها و ضخامت را تعیین می کنیم.

```
import plotly.express as px
```

```
# Create the 3D scatter plot
fig = px.scatter_3d(Dataset,
x="Petal Length",
y="Petal Width",
z="Sepal Width",
color="Iris Type",  # Different colors for each species
color_discrete_sequence=px.colors.sequential.Magma,
title="3D Scatter Plot of Iris Dataset")

# Customize markers
fig.update_traces(marker=dict(size=5,  # Marker size
opacity=0.8,  # Transparency
line=dict(width=10, color='black')),  # Black border around each point
selector=dict(mode='markers'))

# Add interactive layout settings
fig.update_layout(margin=dict(l=0, r=0, b=0, t=40),  # Remove margins
scene=dict(aspectmode='cube'))  # Equal scaling on all axes

# Show plot
fig.show()
```



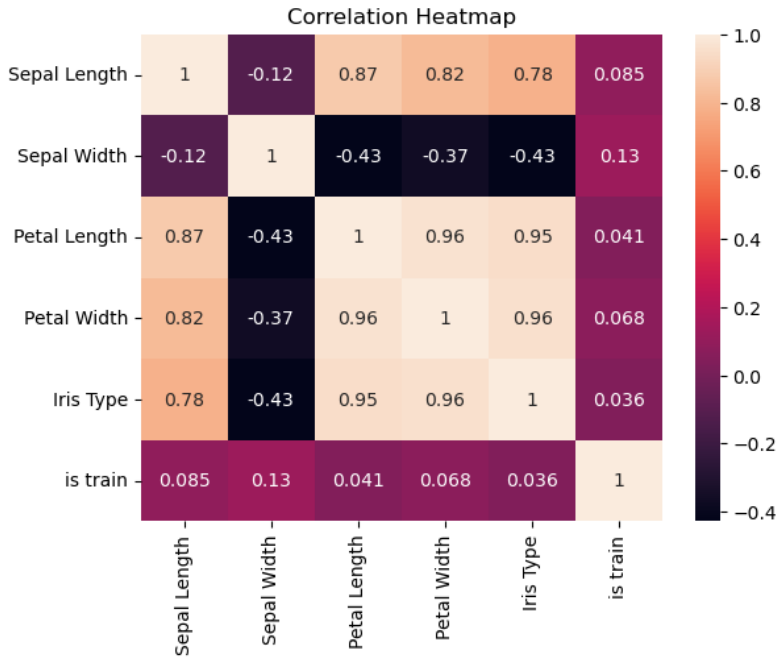

شکل ۱۰.۱: توزیع داده های سه وِیژگی سیستم




شکل ۱۱.۱: توزیع داده ها با سه ویژگی دیگر

در بخش بعد، برای بررسی شباهت فیچر های دیتاست با یکدیگر، ابتدا مقدار correlation آنها را به دست آورده و سپس نقشه حرارتی آن را رسم می کنیم.

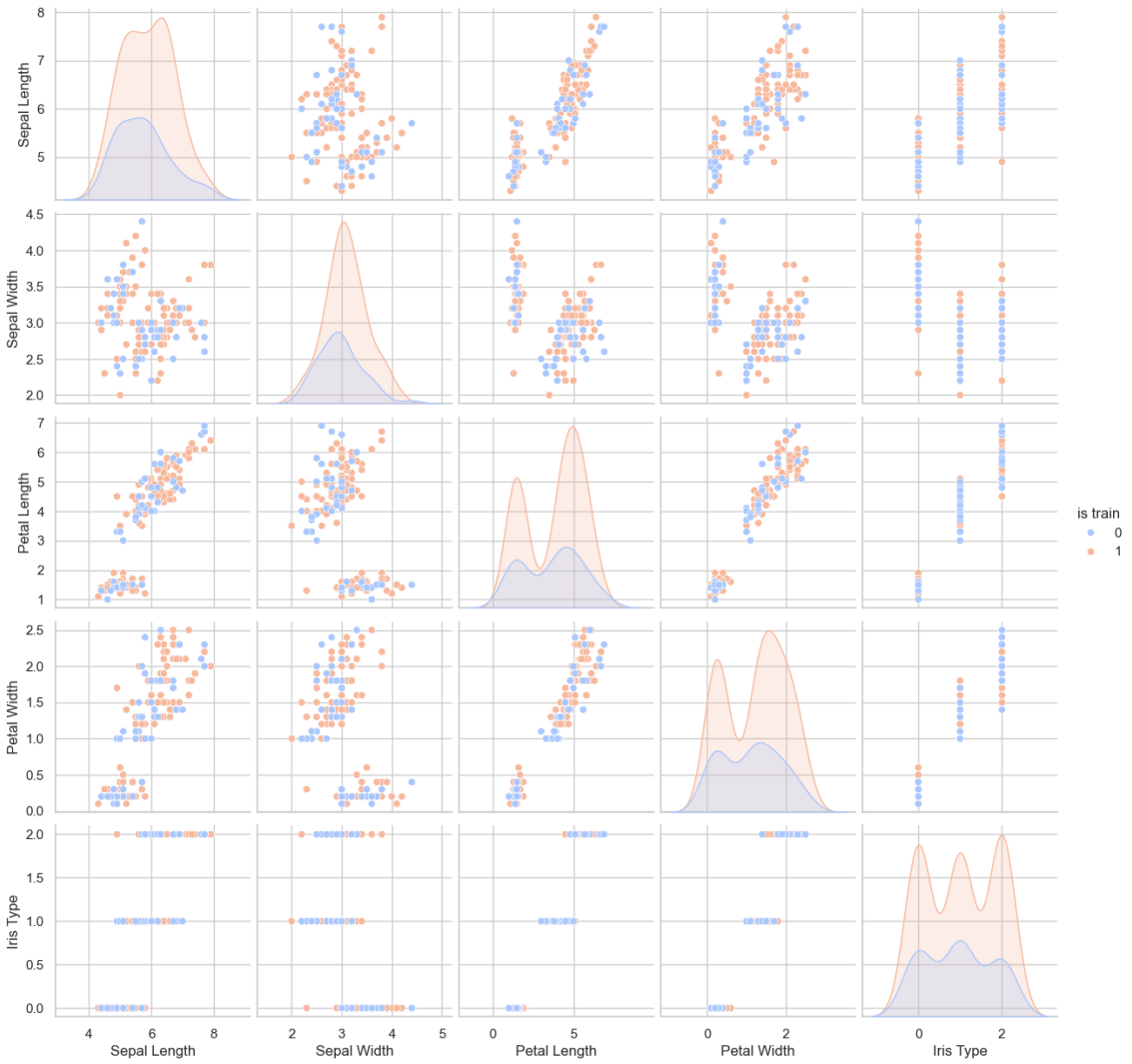
```
sns.heatmap(Dataset.corr(),annot=True )
plt.title("Correlation Heatmap")
```




شکل ۱۲.۱: نمودار حرارتی correlation

در بخش بعد، به نمایش توزیع داده های هر فیچر به تفکیک داده های آموزش و تست می پردازیم. برای به دست آوردن این نمودارها، از دستور pairplot استفاده می شود. با اجرای این دستور برای دیتاست، توزیع دو به دوی تمامی فیچر ها نمایش داده می شود که بر روی قطر اصلی، توزیع هر ویژگی قابل مشاهده است.

```
sns.pairplot(Dataset,hue='is train',palette='coolwarm')
```




شکل ۱۳.۱: $PairPlot$

## ۳.۴.۱ گسسته سازی

در این بخش، ویژگی $petallength$ را گسسته می کنیم. برای این کار، تعداد دسته های مورد نیاز و عنوان آنها را تعریف کرده و سپس با استفاده از دستور cut از کتابخانه ی pandas، مقادی را در بازه های مورد نظر دسته بندی می کنیم.

```
bins = 3  # Number of categories
labels = ["Short", "Medium", "Tall"]  # Custom labels
```

```
Dataset["Discrit Petal Lemgth"]=pd.cut(Dataset["Petal Length"], bins=bins, labels
```

در این صورت، دیتاست نهایی به صورت زیر به دست می آید.

| | Sepal Length | Sepal Width | Petal Length | Petal Width | Iris Type | is train | Discrit Petal Lemgth |
|---|---|---|---|---|---|---|---|
| 0 | 5.7 | 2.8 | 4.5 | 1.3 | 1 | 1 | Medium |
| 1 | 5.5 | 2.6 | 4.4 | 1.2 | 1 | 1 | Medium |
| 2 | 5.7 | 2.6 | 3.5 | 1.0 | 1 | 1 | Medium |
| 3 | 6.7 | 3.1 | 4.4 | 1.4 | 1 | 1 | Medium |
| 4 | 6.3 | 3.3 | 4.7 | 1.6 | 1 | 1 | Medium |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 145 | 5.6 | 2.5 | 3.9 | 1.1 | 1 | 0 | Medium |
| 146 | 5.8 | 2.7 | 5.1 | 1.9 | 2 | 0 | Tall |
| 147 | 5.5 | 2.4 | 3.8 | 1.1 | 1 | 0 | Medium |
| 148 | 6.7 | 3.0 | 5.0 | 1.7 | 1 | 0 | Tall |
| 149 | 4.6 | 3.6 | 1.0 | 0.2 | 0 | 0 | Short |

150 rows × 7 columns

شکل ۱۴.۱: دیتاست پس از گسسته سازی

### ۴.۴.۱ تحلیل آماری

در گام آخر، دیتاست را با استفاده از متد ،describe توصیف کرده و پارامتر های آماری آن را محاسبه می کنیم. در اینجا تنها شرط آن را قرار می دهیم که تنها داده هایی با $IrisType$ برابر با ۰ که معادل Setosa است آورده شود.

```
Dataset[Dataset["Iris Type"] ==0].describe()
```

| | Sepal Length | Sepal Width | Petal Length | Petal Width | Iris Type | is train |
|---|---|---|---|---|---|---|
| count | 50.00000 | 50.000000 | 50.000000 | 50.000000 | 50.0 | 50.00000 |
| mean | 5.00600 | 3.428000 | 1.462000 | 0.246000 | 0.0 | 0.70000 |
| std | 0.35249 | 0.379064 | 0.173664 | 0.105386 | 0.0 | 0.46291 |
| min | 4.30000 | 2.300000 | 1.000000 | 0.100000 | 0.0 | 0.00000 |
| 25% | 4.80000 | 3.200000 | 1.400000 | 0.200000 | 0.0 | 0.00000 |
| 50% | 5.00000 | 3.400000 | 1.500000 | 0.200000 | 0.0 | 1.00000 |
| 75% | 5.20000 | 3.675000 | 1.575000 | 0.300000 | 0.0 | 1.00000 |
| max | 5.80000 | 4.400000 | 1.900000 | 0.600000 | 0.0 | 1.00000 |

شکل ۱۵.۱: تحلیل آماری داده های ستوسا