Capítulo 2

Iniciando o estudo e exploração de dados

Esta parte do material vai ilustrar como a linguagem Python pode ser utilizada para uma análise exploratória de um conjunto de dados, de forma a encontrar padrões nos dados e extrair conhecimento desses dados. Existem duas maneiras de explorar um conjunto de dados: por meio de técnicas estatísticas, mais especificamente da estatística descritiva, e de técnicas de visualização.

Nesta parte do material, serão apresentadas algumas das principais medidas estatísticas utilizadas para descrever um conjunto de dados, bem como suas funções correspondentes nas bibliotecas da linguagem Python consideradas, além de técnicas simples capazes de ilustrar graficamente padrões presentes em um conjunto de dados.

Inicialmente, na Seção 2.1, será apresentada uma breve introdução às operações matemáticas básicas para a manipulação de dados com a biblioteca NumPy. Na Seção 2.2, será discutido como as bases de dados consideradas no presente material estão normalmente estruturadas, e serão apresentadas algumas medidas estatísticas para análise exploratória de dados. Na Seção 2.3, será discutida a ocorrência de dados multivariados, que condizem com os cenários estudados em mineração de dados. Na Seção 2.4, serão demonstrados alguns exemplos simples de gráficos que podem auxiliar em uma análise inicial dos dados. Por fim, na Seção 2.5, serão propostos alguns exercicios com a finalidade de praticar os conceitos apresentados.

2.1. Introdução à NumPy

NumPy é um pacote da linguagem Python que foi desenvolvido para computação científica, área que utiliza computação para resolver problemas complexos. Esse pacote, que será muito utilizado para as análises e experimentos destas notas, possui várias funções que permitem manipular e descrever dados. Este capítulo irá inicialmente mostrar como funções desse pacote podem ser aplicadas a *arrays*, que consistem no tipo básico definido pelo pacote.

2.1.1. *Arrays*

O principal tipo de dados disponibilizado pela biblioteca NumPy e que será mais utilizado no decorrer deste material é conhecido como `numpy.array`. Um *array* pode ser instanciado por meio da chamada `numpy.array(lista)`, na qual `lista` é um objeto do tipo `list` contendo apenas valores numéricos. Por exemplo:

```
import numpy
lista = [1, 2, 3, 4, 5]
x = numpy.array(lista)
print(x)
```

O tipo *array* fornece facilidades para a realização de operações matemáticas com escalares. Para fins de ilustração, execute o código abaixo no terminal do Python:

```
import numpy

x = numpy.array([1, 2, 3, 4, 5])
y = 2

print(x + y)

print(x - y)
print(y - x)

print(x * y)

print(x / y)
print(y / x)
```

Ao inspecionar os resultados do código anterior, é possível observar que nas operações de soma, subtração, multiplicação e divisão em que um dos operandos é um escalar e o outro é um *array*, cada operação ocorre entre o escalar e cada elemento do *array*.

A NumPy também possui funções para realizar operações utilizando dois *arrays* de mesmo tamanho[1]. Como exemplo, execute o código abaixo no terminal do Python:

```
import numpy

x = numpy.array([1, 2, 3, 4, 5])
y = numpy.array([6, 7, 8, 9, 10])

print(x + y)

print(x - y)
print(y - x)

print(x * y)
```

[1]Na verdade, diversas operações podem ser feitas também com *arrays* de tamanhos e dimensões distintas. Entretanto, o comportamento da NumPy em tais cenários pode não ser tão intuitivo. Para o leitor interessado neste tópico, recomenda-se a leitura de: `https://scipy.github.io/old-wiki/pages/EricsBroadcastingDoc`.

```
print(x / y)
print(y / x)
```

Ao analisar as saídas do código anterior, pode-se perceber que, quando os operandos são dois *arrays* do mesmo tamanho, as operações de soma, subtração, multiplicação e divisão ocorrem elemento a elemento. Embora os *arrays* utilizados nos exemplos anteriores tenham apenas uma dimensão, as operações podem ser aplicadas a *arrays* e matrizes com qualquer número de dimensões.

Por fim, a NumPy fornece ainda várias funções pré-definidas para explorar diversas propriedades de um *array* qualquer `x`. Dentre as funções que serão utilizadas ao longo deste material, pode-se mencionar:

- `numpy.sum(x)`: retorna a soma de todos os elementos de `x`.
- `numpy.max(x)`: retorna o valor máximo contido em `x`.
- `numpy.min(x)`: retorna o valor mínimo contido em `x`.

2.2. Exploração de dados

Tipicamente, para boa parte dos problemas em que técnicas mineração de dados e aprendizado de máquina são aplicadas, o conjunto de dados coletados, também conhecido como base de dados, será representado por meio de uma matriz ou tabela, denotada por $X^{n\times m}$, em que n indica o número de objetos observados e m indica o número de características (ou atributos) que foram coletadas para cada objeto.

Como exemplo a ser utilizado no decorrer deste capítulo, considere a base de dados Iris, frequentemente utilizada em livros de mineração de dados e de aprendizado de máquina, originalmente proposta em [Fisher 1936]. Nesta base de dados constam as informações de 150 exemplos observados de plantas provenientes de três diferentes espécies do gênero Iris: *Iris setosa*, *Iris versicolor* e *Iris virginica*. As características coletadas para cada exemplo foram: largura da sépala (cm), comprimento da sépala (cm), largura da pétala (cm) e comprimento da pétala (cm). A Tabela 2.1 apresenta duas observações (exemplos) de cada espécie (classe).

Tabela 2.1: Dois exemplos de cada classe da base de dados Iris.

Largura sépala	Comprimento sépala	Largura pétala	Comprimento pétala	Espécie (classe)
5,1	3,5	1,4	0,2	*Iris setosa*
4,9	3,0	1,4	0,2	*Iris setosa*
7,0	3,2	4,7	1,4	*Iris versicolor*
6,4	3,2	4,5	1,5	*Iris versicolor*
6,3	3,3	6,0	2,5	*Iris virginica*
5,8	2,7	5,1	1,9	*Iris virginica*

A seguir, serão discutidas algumas das principais medidas estatísticas que podem ser utilizadas para uma exploração inicial dos dados com a finalidade de extrair informações relevantes de uma base de dados tal como a Iris.

2.2.1. Dados univariados

Dados univariados são valores de apenas uma variável. No caso de Mineração de Dados (MD), seriam os valores de um único atributo.

2.2.1.1. Medidas de posição ou localidade

2.2.1.1.1 Média

Considerando a existência de n observações diferentes para uma variável qualquer x, a média é definida como:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \tag{1}$$

Na biblioteca NumPy, assumindo que um conjunto de n observações seja representado por um *array* de tamanho n, a média pode ser calculada conforme o exemplo a seguir:

```
import numpy

# gerando um array com 100 valores aleatorios
# sorteados entre 0 e 1000
x = numpy.random.randint(low=0, high=100, size=100)

# calculando a media utilizando os metodos
# numpy.sum e len
media = numpy.sum(x) / len(x)

print(media)

# mais facilmente, podemos utilizar o metodo numpy.mean
media = numpy.mean(x)

print(media)
```

2.2.1.1.2 Mediana

Considerando a existência de n observações para uma variável qualquer x, a mediana pode ser definida como o valor central das observações ordenadas quando n for ímpar. Para os casos em que n for par, a mediana é definida como valor médio entre as duas observações centrais ordenadas [Faceli et al. 2011]. Em Python, a mediana pode ser calculada tal como apresentado no exemplo abaixo:

```
mediana = numpy.median(x)
print(mediana)
```

2.2.1.1.3 Percentis

Um percentil ou quantil, denotado por $p_{y\%}$, representa, dentre m observações, o valor tal que $y\%$ das observações são menores do que ele [Magalhães e de Lima 2000, Faceli et al. 2011]. Esta medida trata-se de uma generalização da mediana (que corresponde a $p_{50\%}$) e do primeiro e terceiro quartis (que correspondem a $p_{25\%}$ e $p_{75\%}$, respectivamente). Para o seu cálculo, a biblioteca NumPy dispõe do seguinte método:

```
# para calcular p15%
percentil_15 = numpy.percentile(x, 15)
print(percentil_15)
```

2.2.1.2. Medidas de dispersão ou espalhamento

2.2.1.2.1 Intervalo ou amplitude

O intervalo (também conhecido como amplitude) consiste na diferença entre o valor máximo e mínimo de um conjunto de observações. Em Python, o mesmo pode ser calculado como:

```
valor_maximo = numpy.max(x)
valor_minimo = numpy.min(x)
intervalo = valor_maximo - valor_minimo
print(intervalo)
```

2.2.1.2.2 Variância e desvio-padrão

A variância de um conjunto de observações, denotada por s^2, é definida como:

$$s^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{2}$$

em que $\bar{x}$ consiste na média das observações (Equação 1). Para manter a mesma unidade dos dados originais, o desvio-padrão é definido como [Magalhães e de Lima 2000]:

$$s = \sqrt{s^2}. \tag{3}$$

O cálculo dessas duas medidas pode ser implementado como:

```
media = numpy.mean(x)
n = len(x)
diferencas = x - media
variancia = numpy.sum(diferencas * diferencas) / (n - 1)
desvio_padrao = numpy.sqrt(variancia)

# de maneira mais facil, podemos utilizar as
```

```
# seguintes funcoes
variancia = numpy.var(x, ddof=1)
desvio_padrao = numpy.std(x, ddof=1)
```

em que o parâmetro `ddof` indica o número de graus de liberdade utilizados. Para o cálculo na Equação (2), as funções `numpy.var` e `numpy.std` utilizam como divisor a fórmula $n - \texttt{ddof}$. Em todos os cenários estudados neste material será utilizada a variância amostral e o desvio-padrão amostral. Portanto, o valor considerado para `ddof` sempre será 1[2].

2.2.1.3. Medidas de distribuição

2.2.1.3.1 Obliquidade

A obliquidade (ou *skewness*, em inglês) trata-se de uma medida de assimetria de uma distribuição de probabilidade em torno de sua média. Ela pode assumir valores negativos, positivos ou próximos de 0. No primeiro caso, a cauda da distribuição é mais alongada à esquerda e, por consequência, a distribuição dos dados concentra-se mais à direita no seu respectivo gráfico. No segundo caso, a cauda da distribuição é mais alongada para a esquerda, o que aponta uma maior concentração dos dados à direita do seu respectivo gráfico. Por fim, no terceiro caso, a distribuição possui caudas aproximadamente balanceadas e, como resultado, ela terá uma maior simetria. Na Figura 2.1 são apresentados exemplos para as duas primeiras situações, quando a variável assume valores contínuos.

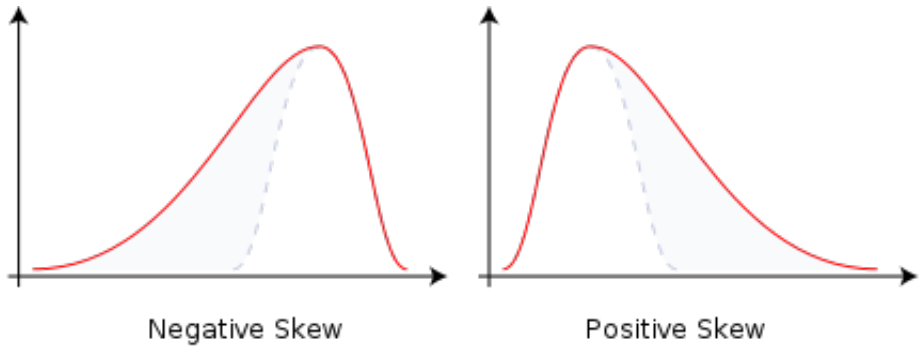

Figura 2.1: Ilustração de obliquidade negativa e positiva (Fonte: [Hermans 2008][3]).

O cálculo da obliquidade é definido por:

$$\text{obliquidade} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{(n-1)\cdot s^3} \tag{4}$$

sendo $\bar{x}$ e s definidos tal como nas equações (1) e (3), nessa ordem.

Em Python, a obliquidade dos dados contidos em um *array* pode ser calculada por meio da biblioteca SciPy da seguinte maneira:

[2] `https://en.wikipedia.org/wiki/Degrees_of_freedom_(statistics)`

[3] A imagem original está disponível sob a licença CC-BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0/`).

```
import numpy
import scipy.stats

# gerando uma amostra com 10000 observacoes a partir de
# uma distribuicao normal com media zero e desvio-padrao
# unitario
dados = numpy.random.normal(loc=0.0, scale=1.0, size=10000)

# como os dados foram gerados segundo uma distribuicao
# normal, que eh simetrica, a obliquidade devera resultar
# em algum valor proximo de 0
obliquidade = scipy.stats.skew(dados)
print(obliquidade)
```

2.2.1.3.2 Curtose

A curtose é uma medida que caracteriza o achatamento da distribuição dos dados. Assim como a obliquidade, os seus valores podem ser negativos, positivos ou próximos de 0. No primeiro caso, a distribuição é mais achatada e apresenta picos mais baixos e caudas mais leves[4] quando comparada à distribuição normal. No segundo caso, a distribuição dos dados apresenta picos mais elevados e caudas mais pesadas[5] ao se comparar à distribuição normal. Por fim, no último caso, a distribuição dos dados apresenta achatamento e caudas próximas ao que ocorre com a distribuição normal.

A equação para o cálculo da curtose é definida como:

$$\text{curtose} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{(n-1)\cdot s^4} \tag{5}$$

sendo $\bar{x}$ e s definidos, respectivamente, pelas Equações (1) e (3).

Tal como a obliquidade, a curtose pode ser calculada por meio da biblioteca SciPy:

```
import numpy
import scipy.stats

# gerando uma amostra com 10000 observacoes a partir de
# uma distribuicao normal com media zero e desvio-padrao
# unitario
dados = numpy.random.normal(loc=0.0, scale=1.0, size=10000)

# como os dados foram gerados segundo uma distribuicao
# normal, que eh simetrica, a curtose devera resultar
# em algum valor proximo de 0
```

[4]Isto é, valores extremos (*outliers*) tendem a ser pouco frequentes. Como um exemplo de extrema curtose, pode-se citar a distribuição uniforme, a qual não apresenta a ocorrência de *outliers*.

[5]Ou seja, *outliers* tendem a ser mais frequentes.

```
curtose = scipy.stats.kurtosis(dados)
print(curtose)
```

2.3. Dados multivariados

Dados multivariados podem ser definidos como conjuntos de dados em que cada observação consiste em um vetor de características e não apenas um único valor, como nos exemplos apresentados na seção anterior. No caso de MD, cada elemento do vetor corresponde a um atributo do conjunto de dados. Em outras palavras, cada observação i pode ser definida como um vetor $x_i = [x_{i1} \quad x_{i2} \quad \cdots \quad x_{im}]^T$, em que m indica o número de características coletadas para cada observação. Portanto, um conjunto de observações.

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix} \tag{6}$$

corresponderá a um conjunto de dados $X^{n \times m}$ (Seção 2.2).

Adotando essa definição, todos os conceitos discutidos na seção anterior podem ser redefinidos. Desse modo, cada medida de posição ou dispersão pode ser reformulada para calcular como resultado um vetor de comprimento m em que cada posição corresponde ao valor de tal medida para cada atributo [Faceli et al. 2011].

2.3.1. Exemplo: Iris

Nesta seção será brevemente apresentado como algumas das medidas discutidas na Seção 2.2.1 podem ser calculadas para dados multivariados na linguagem Python. Para isso, será utilizado como exemplo o conjunto de dados Iris[6], introduzido na Seção 2.2.

Para carregar o arquivo CSV para o conjunto de dados Iris, recomenda-se a utilização da biblioteca Pandas, a qual dispõe do tipo de dados `DataFrame`, que tem uma estrutura tabular, similar àquela apresentada na Tabela 2.1. Assim, pode-se proceder da seguinte maneira:

```
import pandas

# carregando iris.csv
dados = pandas.read_csv('iris.csv')

# imprimindo os dez primeiros exemplos da base de dados
print(data.head(10))
```

Conforme já mencionado, para dados multivariados, as medidas apresentadas na seção anterior serão calculadas sobre cada atributo do problema a ser tratado. Portanto, como exemplo, considere o seguinte código para a Iris:

[6]Disponível em: `https://raw.githubusercontent.com/pandas-dev/pandas/master/pandas/tests/data/iris.csv`.

```
# calculando a media para todos os atributos
print(dados.mean())

# calculando a mediana para todos os atributos
print(dados.median())

# calculando percentil 15% para todos os atributos
print(dados.quantile(0.15))

# calculando o intervalo para todos os atributos
print(dados.max(numeric_only=True) - dados.min(numeric_only=True))

# calculando a variancia para todos os atributos
print(data.var())

# calculando o desvio-padrao para todos os atributos
# (na biblioteca pandas, ddof=1 por padrao)
print(data.std())

# calculando a obliquidade para todos os atributos
print(data.skew())

# calculando a curtose para todos os atributos
print(data.kurtosis())
```

2.4. Visualização de dados

2.4.1. *Histogramas*

Uma das formas mais simples de ilustrar a distribuição de um conjunto de valores de uma variável é o uso de histogramas. Neste tipo de gráfico tem-se, no eixo horizontal, o conjunto (ou intervalos) de valores observados, enquanto que no eixo vertical, apresenta-se a frequência de ocorrência de cada valor (ou valores dentre de um intervalo) presente na amostra analisada. O pacote NumPy fornece uma função para calcular o histograma, que pode ser vista abaixo. Nessa função, `bins` corresponde ao número de barras verticais. Quando o valor de `bins` é definido como `'auto'`, o número de barras é definido automaticamente. Como exemplo, considere o código a seguir:

```
# calculando o histograma para uma variável
import numpy
from matplotlib import pyplot

# Notas na primeira prova
notas = numpy.array([2, 5, 7, 3, 5, 6, 5, 6, 6, 5, 5, 3])

pyplot.hist(notas, bins='auto')
pyplot.title('Histograma')
```

```
pyplot.ylabel('Frequencia')
pyplot.xlabel('Nota')
pyplot.show()
```

O histograma gerado utilizando os dados do código anterior é ilustrado pela Figura 2.2. As cestas sem barras são os intervalos para os quais não havia nenhum valor.

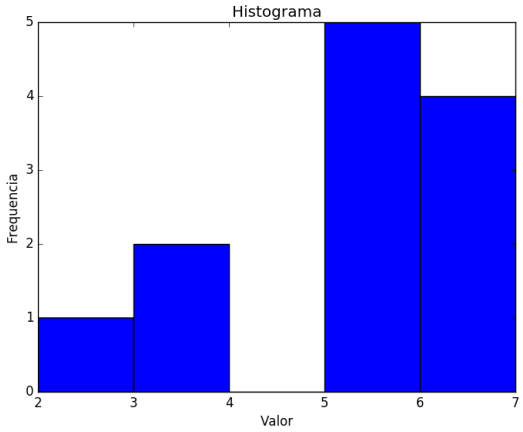

Figura 2.2: Exemplo de histograma para os valores de uma variável.

2.4.2. *Scatter plots*

Um *scatter plot* consiste em um tipo de gráfico comumente utilizado para observar o comportamento entre duas variáveis de uma base de dados. Na Figura 2.3 é apresentado um exemplo de *scatter plot* para a base de dados Iris, em que cada cor indica uma classe diferente.

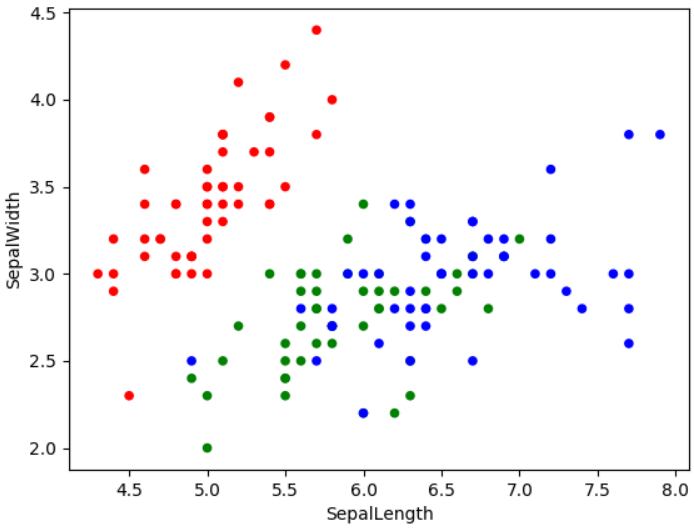

Figura 2.3: Exemplo de *scatter plot* para dois atributos da base de dados Iris.

Um *scatter plot* pode ser gerado por meio do seguinte código:

```
import pandas
from matplotlib import pyplot

# carregando iris.csv
dados = pandas.read_csv('iris.csv')

# criando um dicionario para mapear cada classe para uma cor
classe_cor = {'Iris-setosa' : 'red',
              'Iris-virginica' : 'blue',
             'Iris-versicolor' : 'green'}

# criando uma lista com as cores de cada exemplo
cores = [classe_cor[nome] for nome in dados.Name]

# gerando scatter plot
# no eixo x sera plotado o tamanho da sepala
# no eixo y sera plotado o comprimento da sepala
dados.plot(kind='scatter', x='SepalLength', y='SepalWidth',
           c=cores)

pyplot.show()
```

Alternativamente, pode-se também gerar uma matriz de *scatter plots*, que irá conter os *scatter plots* para todos os pares possíveis de atributos. Ademais, na diagonal de tal matriz, pode-se apresentar informações a respeito de cada atributo (por exemplo, o seu histograma). Na Figura 2.4 é apresentada a matriz de *scatter plots* para a base Iris, com os histogramas dos atributos contidos na diagonal. Tal matriz pode ser gerada por meio do seguinte código:

```
import pandas
from pandas.plotting import scatter_matrix
from matplotlib import pyplot

# carregando iris.csv
dados = pandas.read_csv('iris.csv')

# criando um dicionario para mapear cada classe para uma cor
classe_cor = {'Iris-setosa' : 'red',
              'Iris-virginica' : 'blue',
             'Iris-versicolor' : 'green'}

# criando uma lista com as cores de cada exemplo
cores = [classe_cor[nome] for nome in dados.Name]
```

```
# gerando matriz de scatter plots
scatter_matrix(dados, color=cores)
pyplot.show()
```

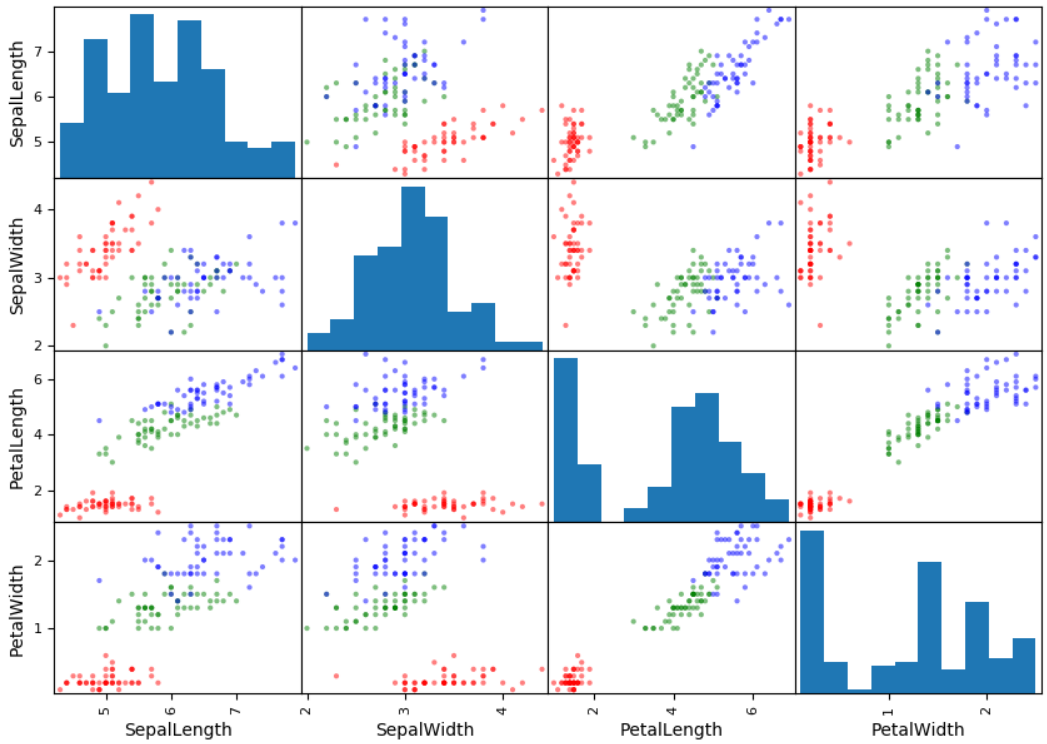

Figura 2.4: Matriz de *scatter plots* para a base de dados Iris.

2.4.3. *Box plots*

Um *box plot* é um gráfico apresentado em formato de caixa, em que a aresta inferior da caixa representa o primeiro quartil (Q_1), a aresta superior representa o terceiro quartil (Q_3) e um traço interno à caixa representa a mediana (Q_2) de uma amostra. Ademais, uma linha tracejada delimita o limite entre o terceiro quartil e o maior valor das observações que é menor ou igual a $Q_3 + 1{,}5 \cdot (Q_3 - Q_1)$ e o limite entre o primeiro quartil e o menor valor na amostra que é maior ou igual a $Q_1 - 1{,}5 \cdot (Q_3 - Q_1)$. Valores abaixo ou acima de tal linha tracejada são representados como círculos, e indicam valores extremos (*outliers*). Na Figura 2.5 é apresentado um exemplo de *box plot* para os atributos da base Iris, o qual pode ser gerado por meio do exemplo de código apresentado a seguir:

```
import pandas
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
```

```
# carregando iris.csv
dados = pandas.read_csv('iris.csv')

# gerando bloxplot para os atributos da Iris
dados.plot(kind='box')
pyplot.show()
```

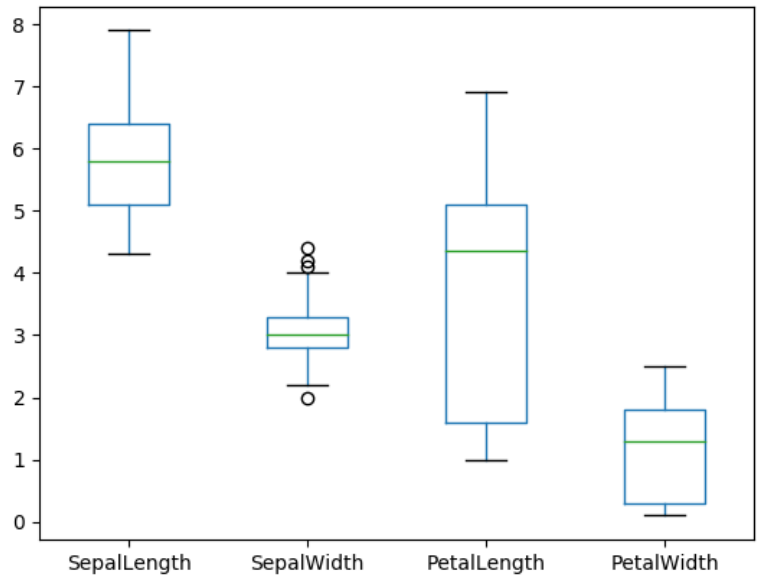

Figura 2.5: *Box plot* para os atributos da base de dados Iris.

2.5. Exercícios

1. Pesquise e explique com as suas palavras sobre o que extraem as medidas de covariância e correlação. Qual a utilidade delas? Aplique-as para a base de dados Iris, apresente e interprete os resultados (dica: use as funções `DataFrame.cov`[7] e `DataFrame.corr`[8]).

2. Considere a matriz de *scatter plots* da Figura 2.4. A partir dos *plots*, o que pode ser observado? Existe alguma classe mais separada das demais? Existem classes sobrepostas? Qual a sua opinião sobre as possíveis implicações das classes sobrepostas para uma tarefa de mineração de dados?

3. Calcule e apresente, para cada atributo da base Iris, sua obliquidade e sua curtose. Compare os resultados obtidos com o *box plot* da Figura 2.5. Ademais, gere os

[7] `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.cov.html`

[8] `https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html`

histogramas para cada atributo da base (dica: pesquise sobre os parâmetros necessários para gerar histogramas com a função `DataFrame.plot`). Que atributos possuem distribuições mais simétricas? Há a presença de valores extremos (*outliers*) para algum atributo? Se sim, qual? O que você acha que pode ser a razão para a ocorrência de tais valores?

Referências

[Faceli et al. 2011] Faceli, K., Lorena, A. C., Gama, J., e Carvalho, A. C. P. L. F. (2011). Inteligência artificial: Uma abordagem de aprendizado de máquina. *Rio de Janeiro: LTC*, 2:192.

[Fisher 1936] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of human genetics*, 7(2):179–188.

[Hermans 2008] Hermans, R. (2008). Negative and positive skew diagrams (English). `https://commons.wikimedia.org/wiki/File:Negative_and_positive_skew_diagrams_(English).svg`. Acesso em: 04 ago. 2017.

[Magalhães e de Lima 2000] Magalhães, M. N. e de Lima, A. C. P. (2000). *Noções de probabilidade e estatística*. IME-USP São Paulo:.