@RAMCHANDRAPADWAL

Practical Guide to Pandas for Data Science

A STEP-BY-STEP GUIDE

Table of Contents

- Introduction to Pandas
  - 1.1 What is Pandas?
  - 1.2 Why Use Pandas?
- Installing and Importing Pandas
  - 2.1 Installation via Anaconda
  - 2.2 Installation via pip
  - 2.3 Importing Pandas
- Data Structures in Pandas
  - 3.1 Series
  - 3.2 DataFrame
  - 3.3 Panel
- Reading and Writing Data
  - 4.1 Reading CSV Files
  - 4.2 Writing CSV Files
  - 4.3 Reading Excel Files
  - 4.4 Writing Excel Files
- Data Manipulation
  - 5.1 Indexing and Selection
  - 5.2 Filtering Data
  - 5.3 Sorting Data
  - 5.4 Aggregating Data
  - 5.5 Handling Missing Data
- Data Cleaning
  - 6.1 Removing Duplicates
  - 6.2 Renaming Columns
  - 6.3 Handling Null Values
  - 6.4 Changing Data Types
- Data Visualization
  - 7.1 Line Plots
  - 7.2 Bar Plots
  - 7.3 Scatter Plots
  - 7.4 Histograms
  - 7.5 Box Plots
- Time Series Analysis
  - 8.1 Creating Time Series Data
  - 8.2 Resampling and Frequency Conversion
  - 8.3 Shifting and Lagging
  - 8.4 Rolling Window Functions
- Conclusion

@RAMCHANDRAPADWAL

CHAPTER N.1

Introduction to Pandas

A Step-by-Step Guide

1.1 WHAT IS PANDAS?

Pandas is an open-source library in Python that provides data manipulation and analysis tools. It is built on top of NumPy and provides easy-to-use data structures and data analysis functions. Pandas is widely used in the field of data science for tasks such as data cleaning, data transformation, data visualization, and data analysis.

1.2 WHY USE PANDAS?

Pandas offers several advantages for data scientists:

- **Easy handling of structured data:** Pandas provides powerful data structures, such as Series and DataFrame, that allow for easy manipulation and analysis of structured data.
- **Data alignment and integration**: Pandas can handle data from various sources and align them based on common indices or column names, making it easy to integrate data from different sources.
- **Efficient data manipulation**: Pandas provides vectorized operations and optimized algorithms, which significantly speed up data manipulation tasks compared to traditional Python methods.
- **Missing data handling**: Pandas offers flexible tools for handling missing data, allowing users to either drop or fill in missing values.
- **Data visualization**: Pandas integrates well with popular data visualization libraries, such as Matplotlib and Seaborn, enabling the creation of insightful visualizations.

CHAPTER N.2

Installing and Importing Pandas

A Step-by-Step Guide

2.1 Installation via Anaconda

If you have Anaconda installed, you can install Pandas by following these steps:

- Open the Anaconda Navigator or Anaconda Prompt.
- Create a new environment (optional but recommended).
- Select the desired environment and click on "Open Terminal" or open the Anaconda Prompt.
- Type the command: conda install pandas and press Enter.

2.2 Installation via pip

If you have Python installed, you can use pip to install Pandas by following these steps:

- Open the command prompt or terminal.
- Type the command: pip install pandas and press Enter.

2.3 Importing Pandas

After installing Pandas, you need to import it into your Python environment before using it in your code. Import Pandas with the following statement:

```python
import pandas as pd
```

The commonly used alias "pd" makes it easier to refer to Pandas functions and objects.

@RAMCHANDRAPADWAL

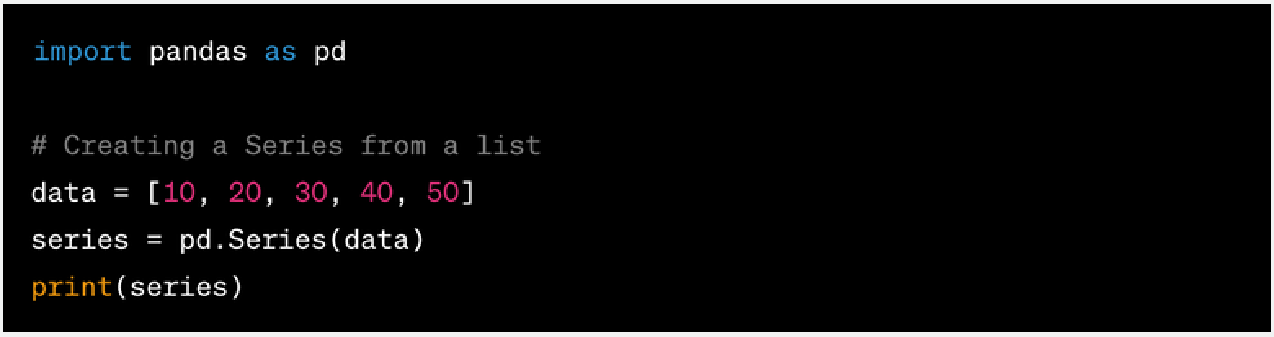
CHAPTER N.3

Data Structures in Pandas

A Step-by-Step Guide

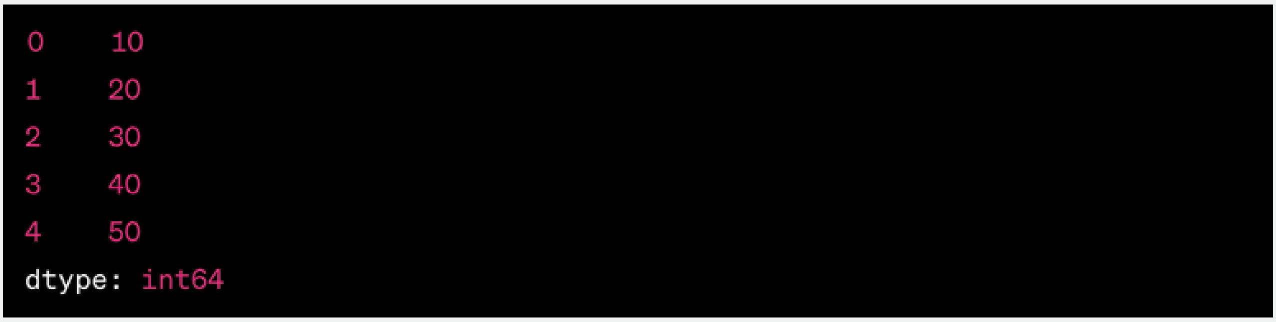
3.1 Series

A Series is a one-dimensional labeled array that can hold any data type. It is similar to a column in a spreadsheet or a one-dimensional NumPy array. You can create a Series using the **pd.Series()** function.



```
import pandas as pd

# Creating a Series from a list
data = [10, 20, 30, 40, 50]
series = pd.Series(data)
print(series)
```

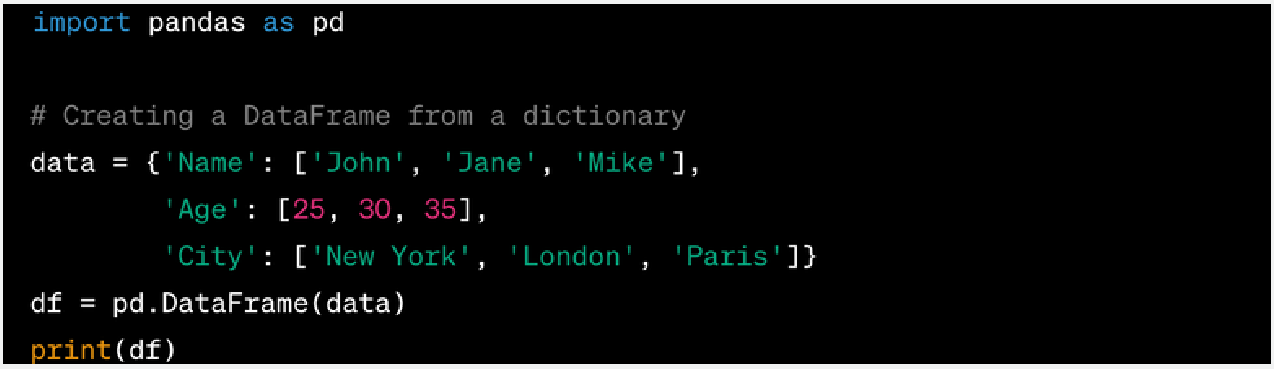
OUTPUT:



```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

3.2 DataFrame

A Data Frame is a two-dimensional labeled data structure with columns of potentially different data types. It is similar to a table in a relational database or a spreadsheet. You can create a DataFrame using the **pd.DataFrame()** function.



```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {'Name': ['John', 'Jane', 'Mike'],
        'Age': [25, 30, 35],
        'City': ['New York', 'London', 'Paris']}
df = pd.DataFrame(data)
print(df)
```

OUTPUT:

```
   Name  Age      City
0  John   25  New York
1  Jane   30    London
2  Mike   35     Paris
```

3.3 Panel

A Panel is a three-dimensional data structure that can hold multiple DataFrames. It is less commonly used compared to Series and DataFrame. You can create a Panel using the **pd.Panel()** function.

```
import pandas as pd
import numpy as np

# Creating a Panel from a three-dimensional NumPy array
data = np.random.rand(3, 4, 5)
panel = pd.Panel(data)
print(panel)
```

OUTPUT:

```
<class 'pandas.core.panel.Panel'>
Dimensions: 3 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 2
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4
```

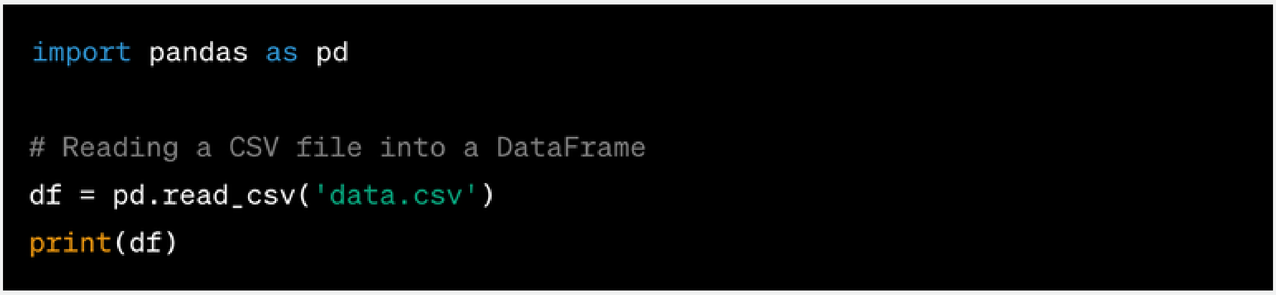
CHAPTER N.4

Reading and Writing Data

A Step-by-Step Guide

4.1 Reading CSV Files

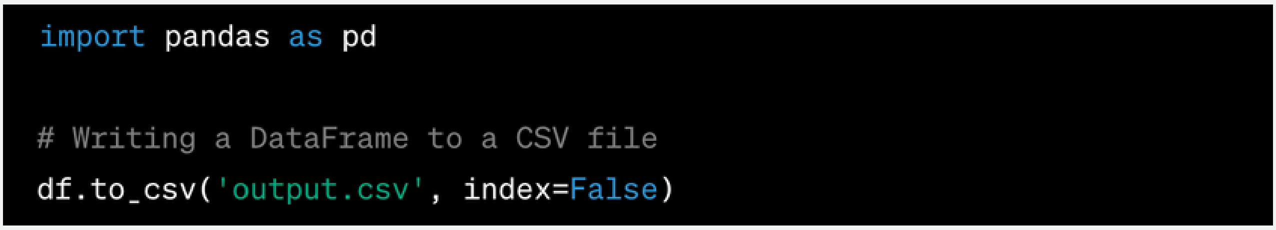
CSV (Comma-Separated Values) files are a common file format for storing tabular data. Pandas provides the **pd.read_csv()** function to read CSV files into a DataFrame.



```
import pandas as pd

# Reading a CSV file into a DataFrame
df = pd.read_csv('data.csv')
print(df)
```

4.2 Writing CSV Files

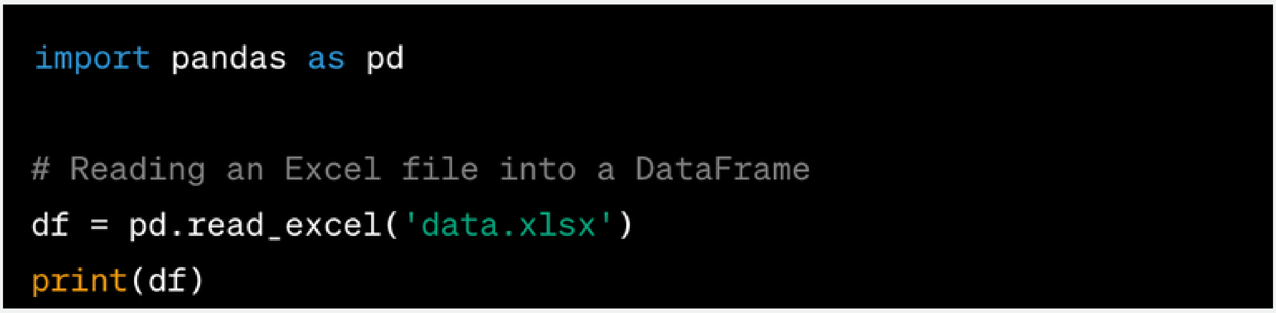
You can save a DataFrame to a CSV file using the **pd.to_csv()** function.



```
import pandas as pd

# Writing a DataFrame to a CSV file
df.to_csv('output.csv', index=False)
```

4.3 Reading Excel Files

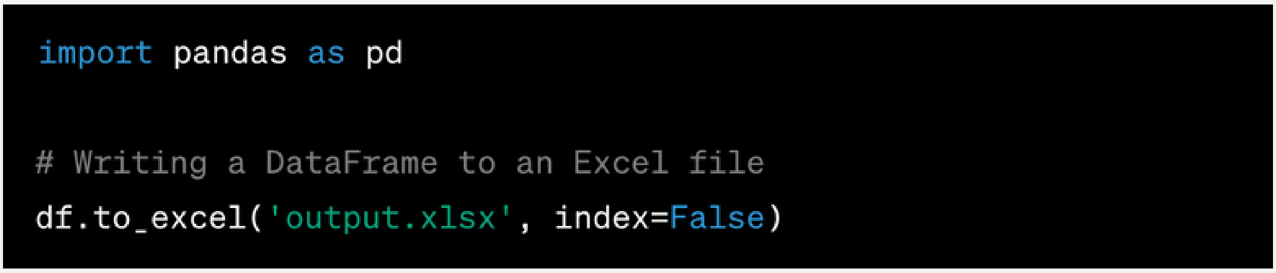
Pandas also supports reading data from Excel files. You can use the **pd.read_excel()** function to read Excel files into a DataFrame.



```
import pandas as pd

# Reading an Excel file into a DataFrame
df = pd.read_excel('data.xlsx')
print(df)
```

4.4 Writing Excel Files

To write a DataFrame to an Excel file, you can use the **pd.to_excel()** function.



```python
import pandas as pd

# Writing a DataFrame to an Excel file
df.to_excel('output.xlsx', index=False)
```

@RAMCHANDRAPADWAL

CHAPTER N.5

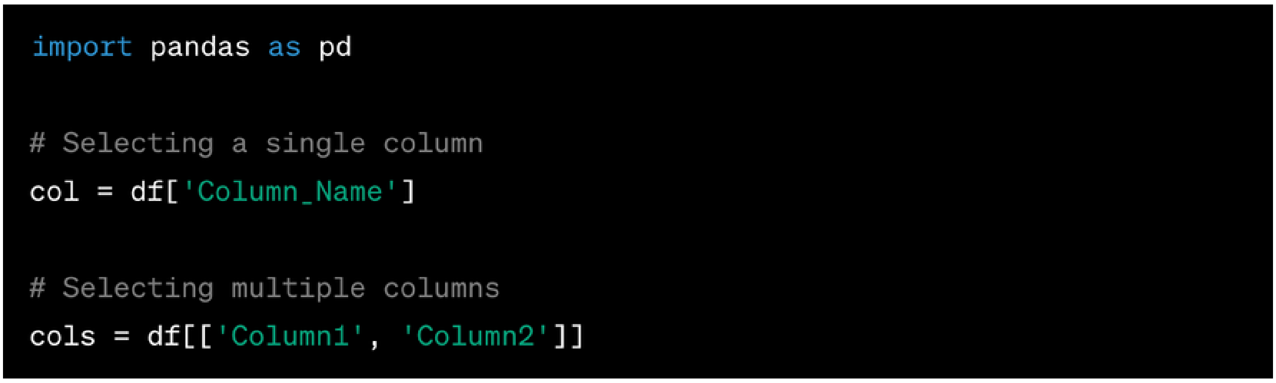
Data Manipulation

A Step-by-Step Guide

5.1ndexing and Selection

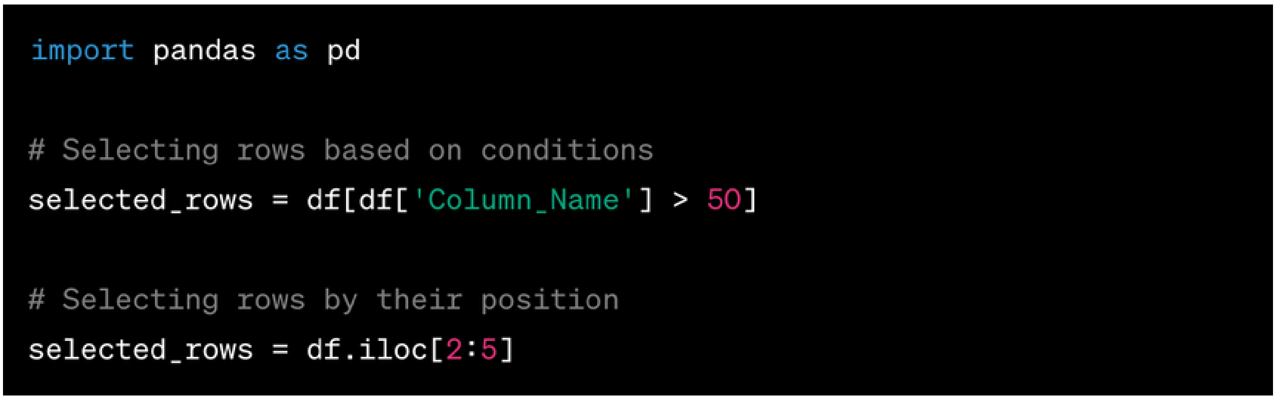
Pandas provides powerful indexing and selection capabilities. You can access and manipulate data in a DataFrame using various indexing techniques.

- **Selecting Columns**: You can select one or more columns from a DataFrame using the column names.



```
import pandas as pd

# Selecting a single column
col = df['Column_Name']

# Selecting multiple columns
cols = df[['Column1', 'Column2']]
```

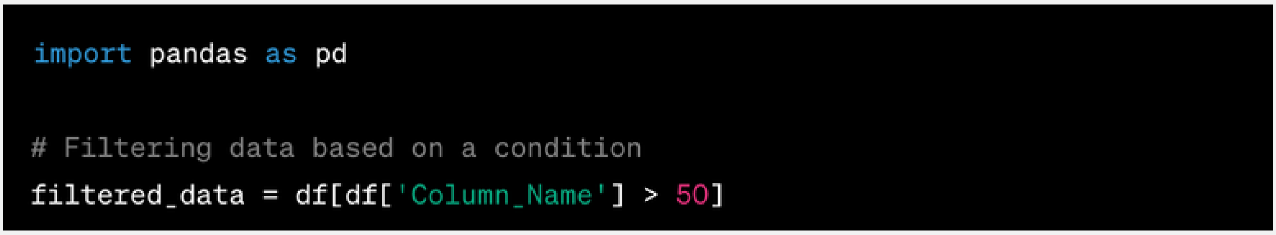
- **Selecting Rows**: You can select rows based on conditions or by their position.



```
import pandas as pd

# Selecting rows based on conditions
selected_rows = df[df['Column_Name'] > 50]

# Selecting rows by their position
selected_rows = df.iloc[2:5]
```

5.2 Filtering Data

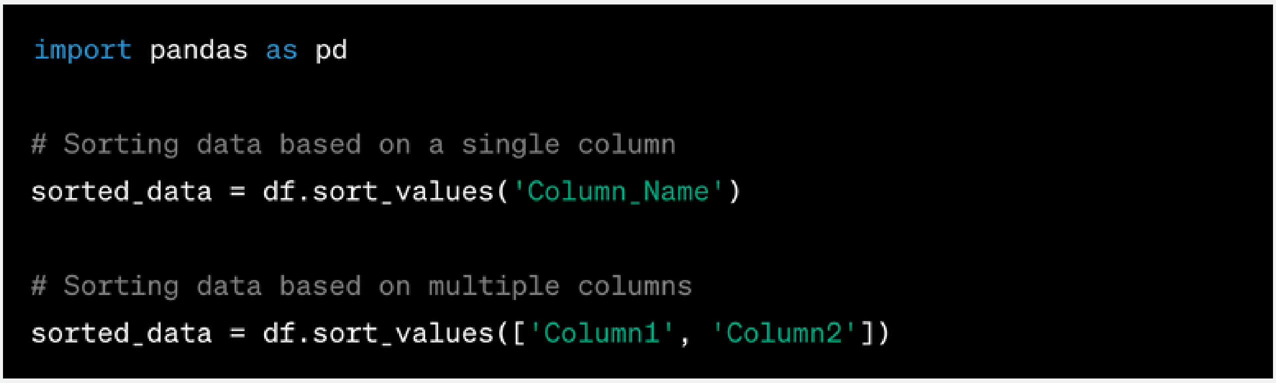
You can filter data in a DataFrame based on specific conditions.



```
import pandas as pd

# Filtering data based on a condition
filtered_data = df[df['Column_Name'] > 50]
```

5.3 Sorting Data

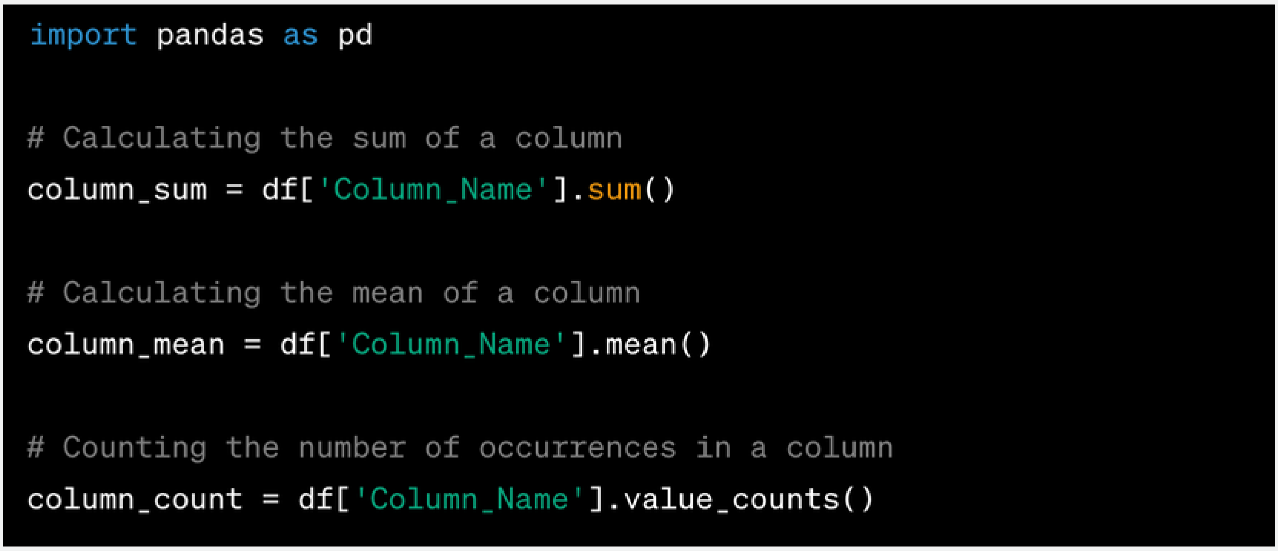
Pandas allows you to sort data based on one or more columns.



```
import pandas as pd

# Sorting data based on a single column
sorted_data = df.sort_values('Column_Name')

# Sorting data based on multiple columns
sorted_data = df.sort_values(['Column1', 'Column2'])
```

5.4 Aggregating Data

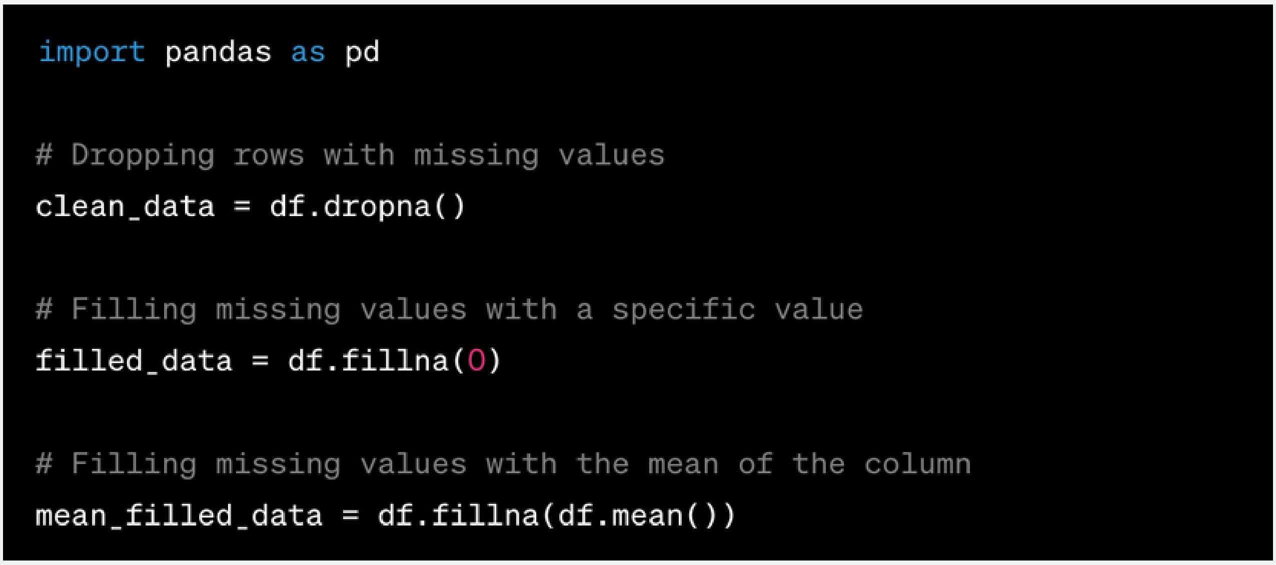
You can perform aggregation operations on your data, such as calculating sums, means, or counts.



```
import pandas as pd

# Calculating the sum of a column
column_sum = df['Column_Name'].sum()

# Calculating the mean of a column
column_mean = df['Column_Name'].mean()

# Counting the number of occurrences in a column
column_count = df['Column_Name'].value_counts()
```

5.5 Handling Missing Data

Pandas provides various methods to handle missing data, such as dropping or filling missing values.



```python
import pandas as pd

# Dropping rows with missing values
clean_data = df.dropna()

# Filling missing values with a specific value
filled_data = df.fillna(0)

# Filling missing values with the mean of the column
mean_filled_data = df.fillna(df.mean())
```

@RAMCHANDRAPADWAL

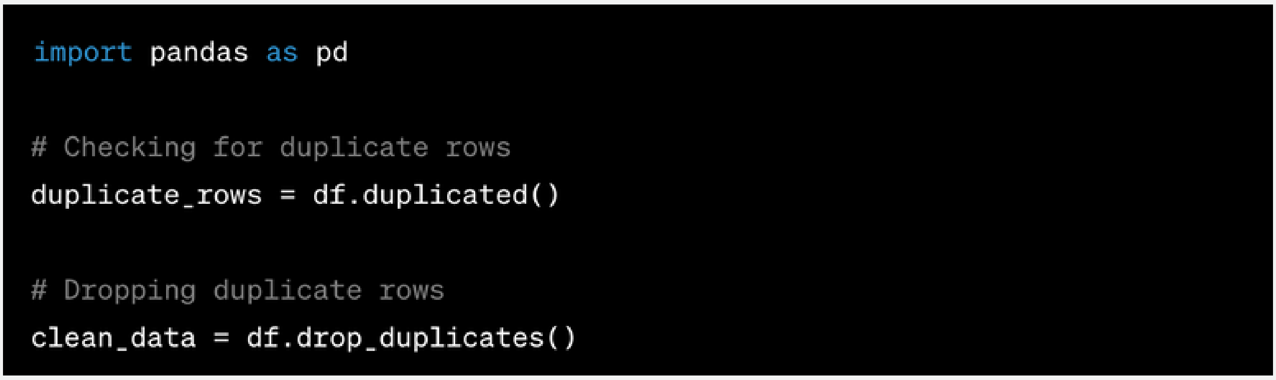
CHAPTER N.6

Data Cleaning

A Step-by-Step Guide

6.1 Removing Duplicates

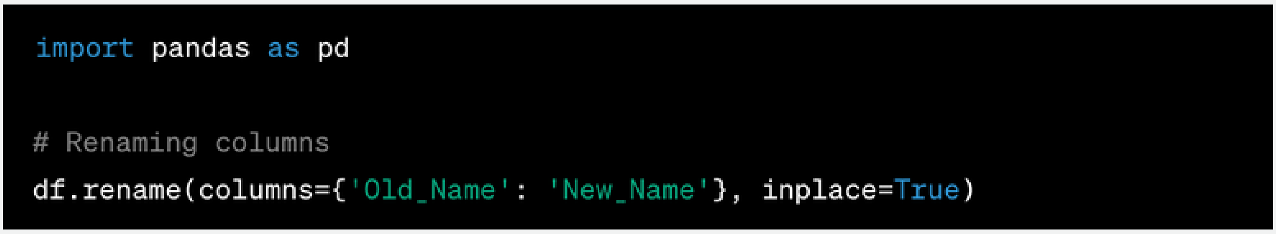
You can remove duplicate rows from a DataFrame using the **duplicated()** and **drop_duplicates()** methods.



```
import pandas as pd

# Checking for duplicate rows
duplicate_rows = df.duplicated()

# Dropping duplicate rows
clean_data = df.drop_duplicates()
```

6.2 Renaming Columns

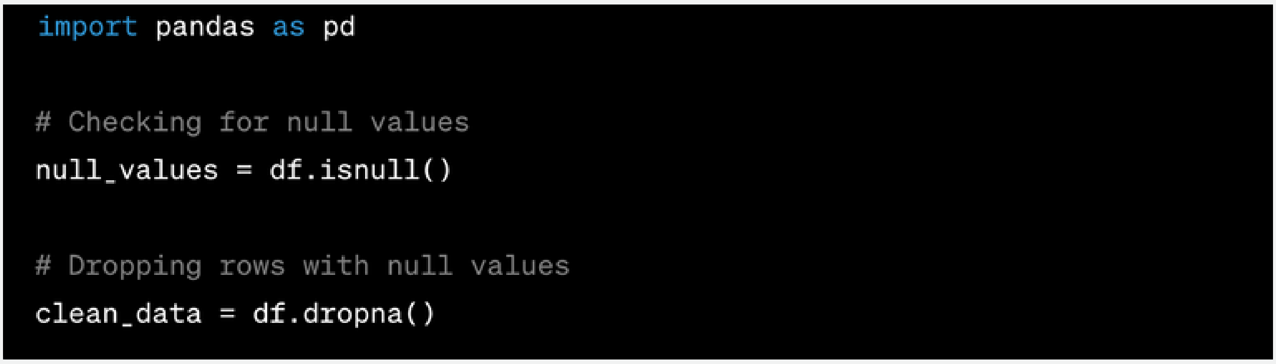
You can rename columns in a DataFrame using the **rename()** method.



```
import pandas as pd

# Renaming columns
df.rename(columns={'Old_Name': 'New_Name'}, inplace=True)
```

6.3 Handling Null Values

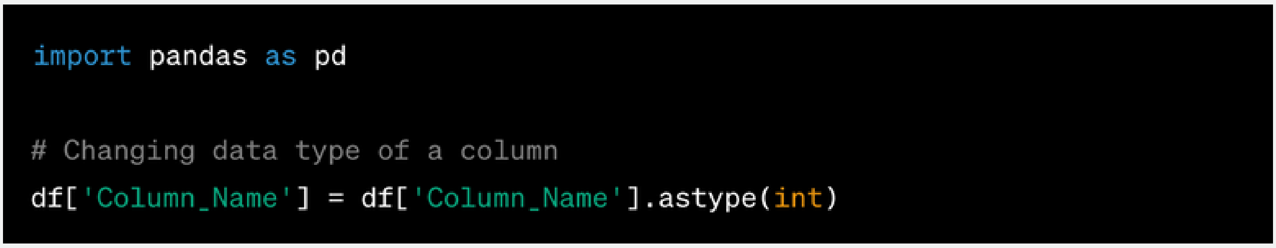
Pandas provides methods to handle null values, such as **isnull()**, **notnull()**, and **dropna()**.



```
import pandas as pd

# Checking for null values
null_values = df.isnull()

# Dropping rows with null values
clean_data = df.dropna()
```

6.4 Changing Data Types

You can change the data type of columns in a DataFrame using the **astype()** method.



```python
import pandas as pd

# Changing data type of a column
df['Column_Name'] = df['Column_Name'].astype(int)
```

@RAMCHANDRAPADWAL

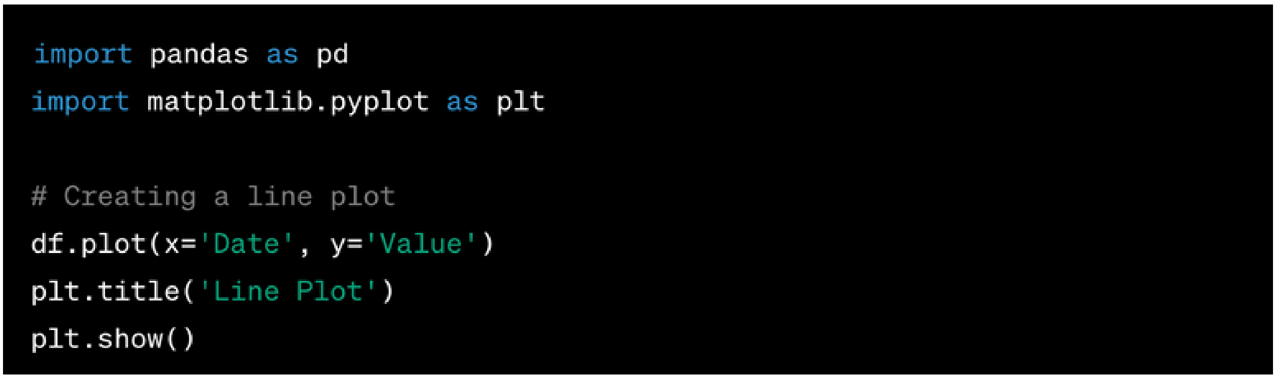
CHAPTER N.7

Data Visualization

A Step-by-Step Guide

7.1 Line Plots

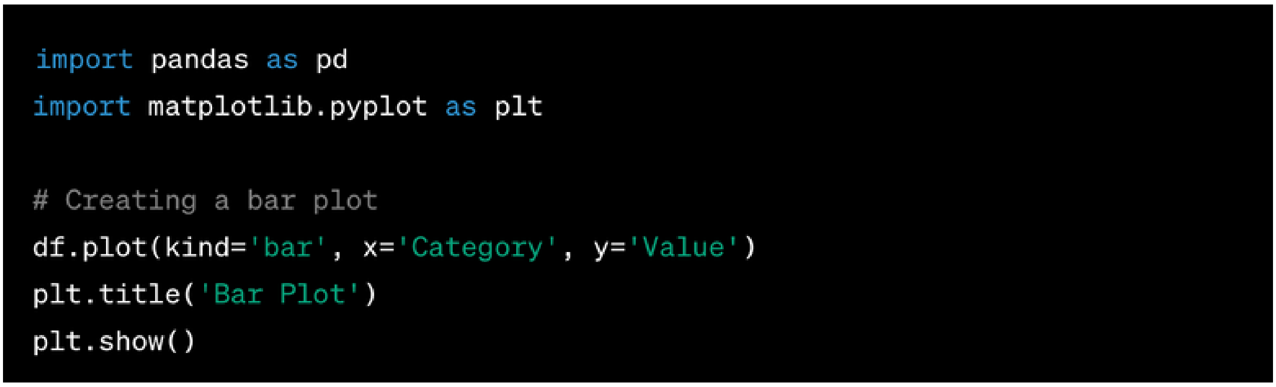
You can create line plots using Pandas and visualize trends in your data.



```
import pandas as pd
import matplotlib.pyplot as plt

# Creating a line plot
df.plot(x='Date', y='Value')
plt.title('Line Plot')
plt.show()
```

7.2 Bar Plots

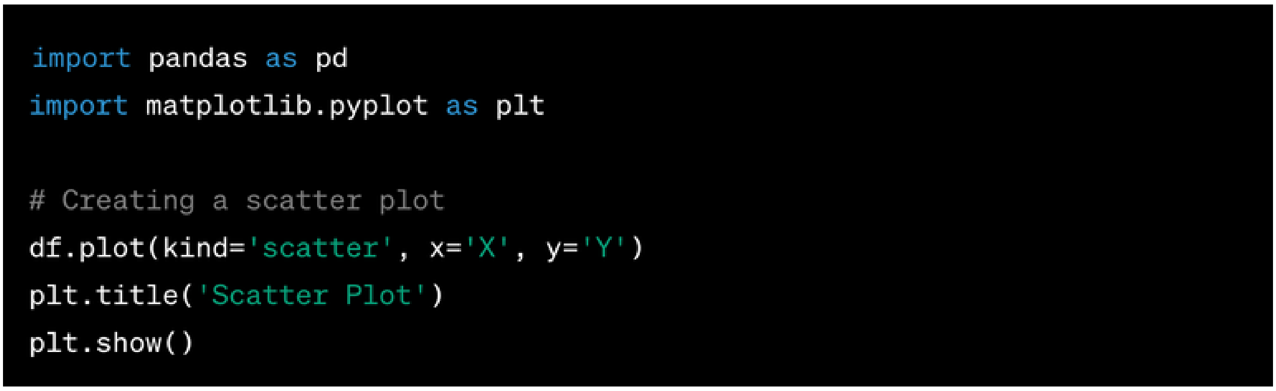
Bar plots are useful for comparing categorical data.



```
import pandas as pd
import matplotlib.pyplot as plt

# Creating a bar plot
df.plot(kind='bar', x='Category', y='Value')
plt.title('Bar Plot')
plt.show()
```

7.3 Scatter Plots

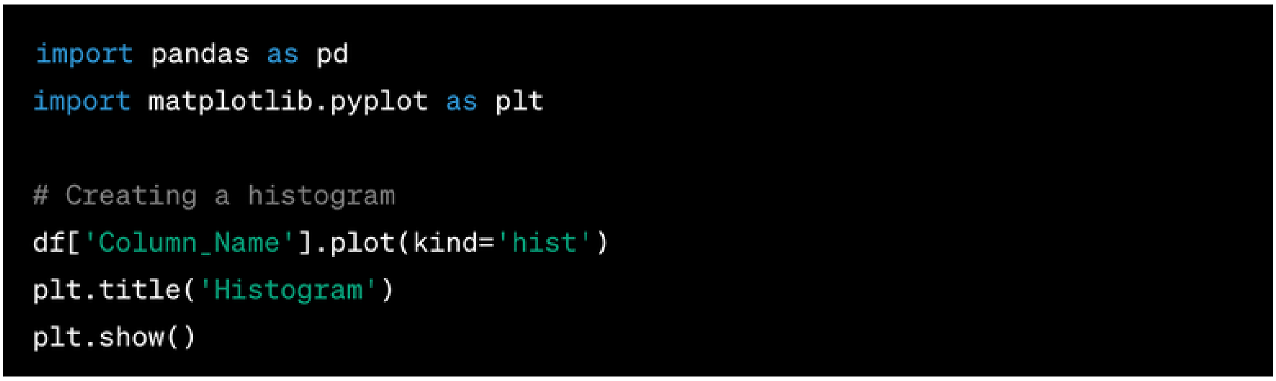
Scatter plots are useful for visualizing the relationship between two variables.



```
import pandas as pd
import matplotlib.pyplot as plt

# Creating a scatter plot
df.plot(kind='scatter', x='X', y='Y')
plt.title('Scatter Plot')
plt.show()
```

7.4 Histograms

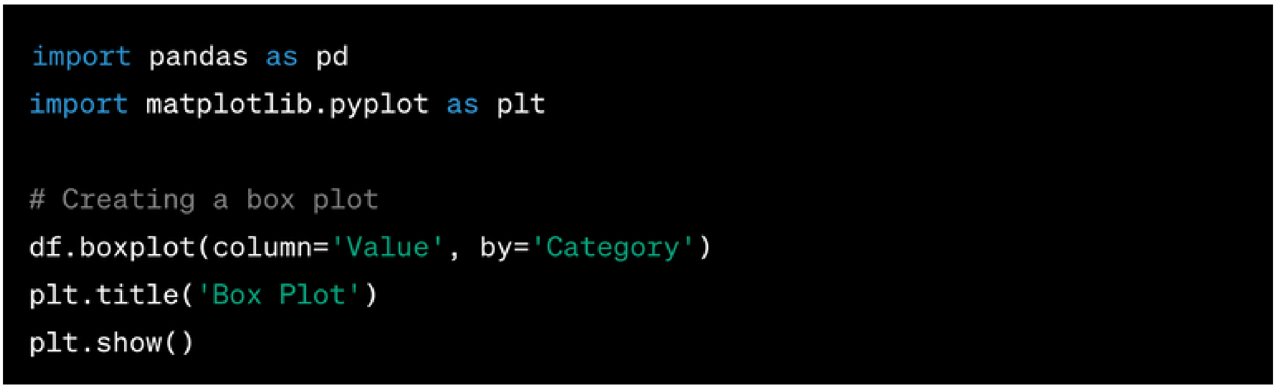
Histograms help visualize the distribution of a single variable.



```python
import pandas as pd
import matplotlib.pyplot as plt

# Creating a histogram
df['Column_Name'].plot(kind='hist')
plt.title('Histogram')
plt.show()
```

7.5 Box Plots

Box plots display the distribution of a variable across different categories.



```python
import pandas as pd
import matplotlib.pyplot as plt

# Creating a box plot
df.boxplot(column='Value', by='Category')
plt.title('Box Plot')
plt.show()
```

@RAMCHANDRAPADWAL

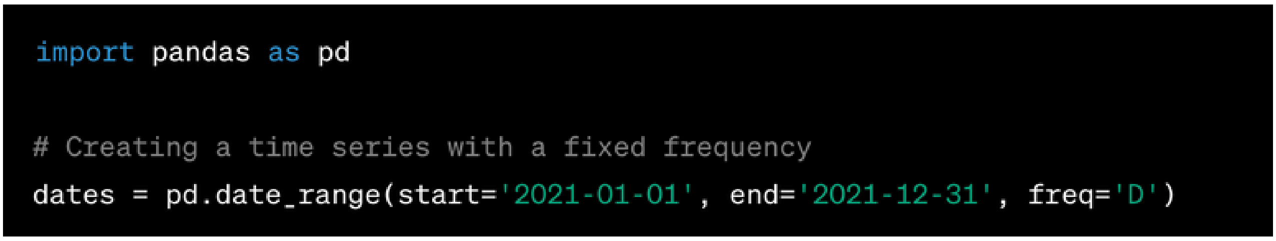
CHAPTER N.8

Time Series Analysis

A Step-by-Step Guide

8.1 Creating Time Series Data

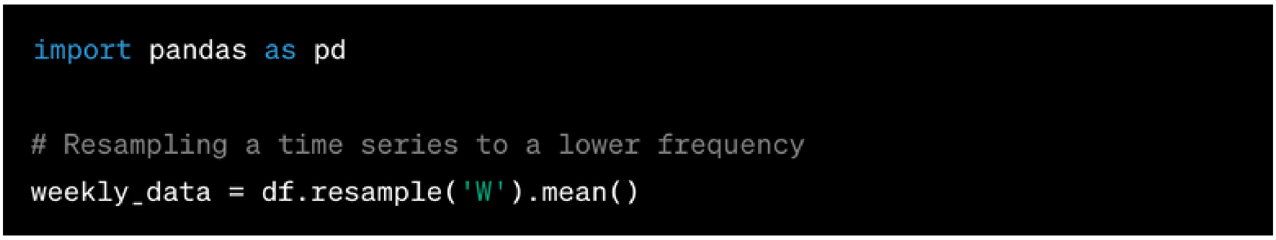
Pandas provides functions to create time series data, such as **date_range()** and **to_datetime()**.



```
import pandas as pd

# Creating a time series with a fixed frequency
dates = pd.date_range(start='2021-01-01', end='2021-12-31', freq='D')
```

8.2 Resampling and Frequency Conversion

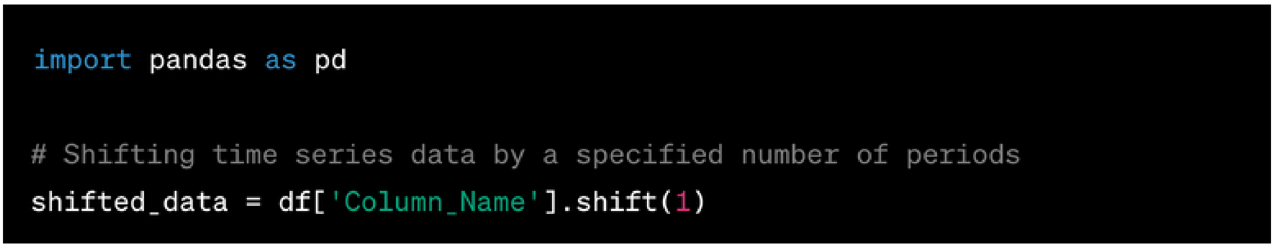
Pandas offers resampling functions to change the frequency of time series data.



```
import pandas as pd

# Resampling a time series to a lower frequency
weekly_data = df.resample('W').mean()
```

8.3 Shifting and Lagging

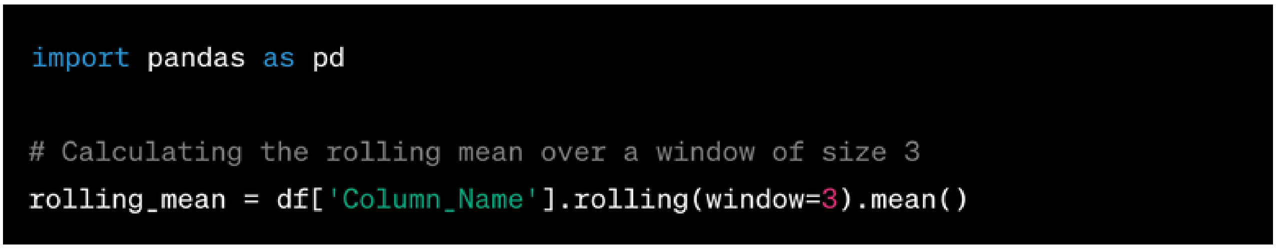
You can shift or lag time series data using the **shift()** method.



```
import pandas as pd

# Shifting time series data by a specified number of periods
shifted_data = df['Column_Name'].shift(1)
```

8.4 Rolling Window Functions

Pandas provides rolling window functions for calculating statistics over a specified window.



```
import pandas as pd

# Calculating the rolling mean over a window of size 3
rolling_mean = df['Column_Name'].rolling(window=3).mean()
```

Conclusion

This practical guide has introduced you to the key features of Pandas for data science. You learned about installing Pandas, importing it into your Python environment, and working with its data structures, such as Series, DataFrame, and Panel. You also explored various data manipulation techniques, data cleaning methods, data visualization options, and time series analysis capabilities in Pandas. With this knowledge, you can start using Pandas effectively for your data science projects.