

Data Structures and Analysis in Python: A Practical Approach

Sabbir Ahmed

February 5, 2025



1 Introduction

This report presents a comprehensive implementation of various Python programming concepts, focusing on data structures, numerical computations, data analysis, and visualization. The implementations demonstrate proficiency in using essential Python libraries such as NumPy, Pandas, and Matplotlib.

[More information on Github repo](#)

2 Basic Python Data Structures

2.1 List Operations



2.1.1 Objective

Implementation of list operations to remove duplicates and sort numbers in ascending order, demonstrating fundamental list manipulation techniques in Python.

2.1.2 Implementation

```
1 numlist = [3, 9, 4, 2, 4, 1, 5, 6, 10, 8, 7, 9]
2
3 res = []
4
5 for i in numlist:
6     if i not in res:
7         res.append(i)
8
9 res.sort()
10 print("result 1: ", res)
11
12 print("result 2: ", sorted(list(set(numlist))))
```

Listing 1: List Operations Implementation

2.1.3 Output

```
wrangling.py Tasks.md list.py Python
```

```
list.py > ...
1 numlist = [3, 9, 4, 2, 4, 1, 5, 6, 10, 8,
2
3 res = []
4
5 for i in numlist:
6     if i not in res:
7         res.append(i)
8
9 res.sort()
10 print("result 1: ", res)
11
12 print("result 2: ", sorted(list(set(numli
```

```
5e80484..d7ec4d8 main -> main
PS C:\Users\msa29\Desktop\Python-Practice> & C:
:/Users/msa29/AppData/Local/Microsoft/WindowsApp
s/python3.11.exe c:/Users/msa29/Desktop/Python-
Practice/list.py
result 1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result 2: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
PS C:\Users\msa29\Desktop\Python-Practice>
```

Figure 1: List

2.2 Set Operations



2.2.1 Objective

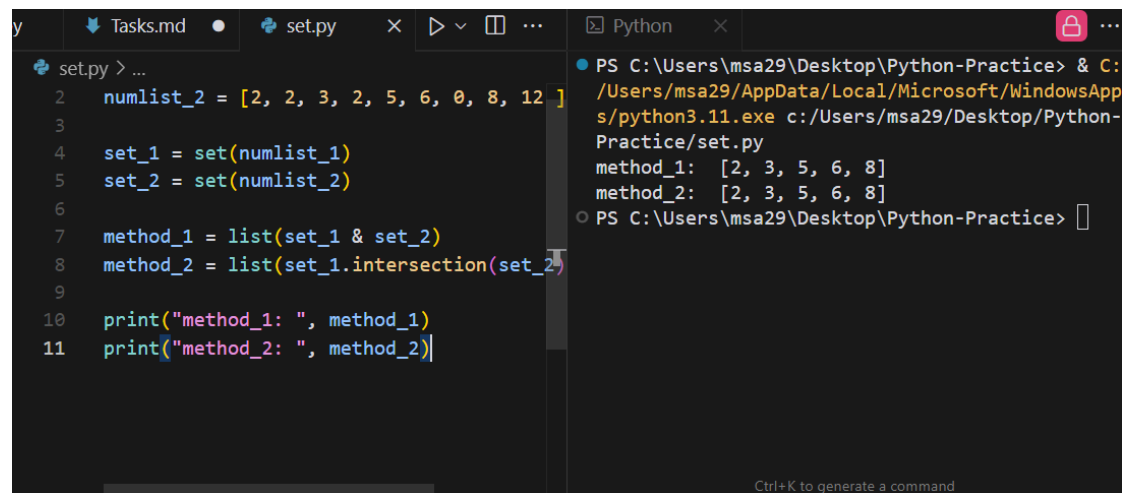
Demonstration of set operations to find common elements between two lists, showcasing the efficiency of set operations in Python.

2.2.2 Implementation

```
1 numlist_1 = [3, 9, 4, 2, 4, 1, 5, 6, 10, 8, 7, 9]
2 numlist_2 = [2, 2, 3, 2, 5, 6, 0, 8, 12 ]
3
4 set_1 = set(numlist_1)
5 set_2 = set(numlist_2)
6
7 method_1 = list(set_1 & set_2)
8 method_2 = list(set_1.intersection(set_2))
9
10 print("method_1: ", method_1)
11 print("method_2: ", method_2)
```

Listing 2: Set Operations Implementation

2.2.3 Output



```
set.py > ...
2 numlist_2 = [2, 2, 3, 2, 5, 6, 0, 8, 12]
3
4 set_1 = set(numlist_1)
5 set_2 = set(numlist_2)
6
7 method_1 = list(set_1 & set_2)
8 method_2 = list(set_1.intersection(set_2))
9
10 print("method_1: ", method_1)
11 print("method_2: ", method_2)]

PS C:\Users\msa29\Desktop\Python-Practice> & C:
/Users/msa29/AppData/Local/Microsoft/WindowsApp
s/python3.11.exe c:/Users/msa29/Desktop/Python-
Practice/set.py
method_1: [2, 3, 5, 6, 8]
method_2: [2, 3, 5, 6, 8]
PS C:\Users\msa29\Desktop\Python-Practice> ]
```

Figure 2: Set

2.3 Tuple Operations



2.3.1 Objective

Creation and manipulation of student records using tuples, demonstrating the immutable nature of tuples and sorting operations.

2.3.2 Implementation

```
1 students = (
2     ("Sabbir", 22, 3.6),
3     ("Jiyon", 23, 3.8),
4     ("Ishaq", 24, 3.7),
5     ("Rakib", 22, 3.5),
6     ("Sakib", 23, 3.6)
7 )
8
9 def student_sort(student):
10     return student[2]
11
12 sorted_students = sorted(students, key=student_sort)
13
14 for student in sorted_students:
15     print(student)
```

Listing 3: Tuple Operations Implementation

2.3.3 Output

```
tuple.py > ...
1  students = (
2      ("Sabbir", 22, 3.6),
3      ("Jiyon", 23, 3.8),
4      ("Ishaq", 24, 3.7),
5      ("Rakib", 22, 3.5),
6      ("Sakib", 23, 3.6)
7  )
8
9  def student_sort(student):
10     return student[2]
11
12  sorted_students = sorted(students, key=student_sort)
13
14  for student in sorted_students:
15     print(student)
16
PS C:\Users\msa29\Desktop\Python-Practice> & C:/Users/msa29/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Python-Practice/tuple.py
('Rakib', 22, 3.5)
('Sabbir', 22, 3.6)
('Sakib', 23, 3.6)
('Ishaq', 24, 3.7)
('Jiyon', 23, 3.8)
PS C:\Users\msa29\Desktop\Python-Practice>
```

Figure 3: Tuple

2.4 Dictionary Operations



2.4.1 Objective

Implementation of a word occurrence counter using dictionaries, showcasing text analysis capabilities.

2.4.2 Implementation

```
1 text = "Hello there, I'm Sabbir Ahmed, a passionate and curious
      individual from Dhaka, Bangladesh. As a Computer Science and
      Engineering student at Green University of Bangladesh, I am
      constantly seeking new opportunities to expand my knowledge and
      skills in the software engineering and data science field."
2
3 words = text.split()
4
5 def word_count(words):
6     word_count = {}
7     for word in words:
8         if word in word_count:
9             word_count[word] += 1
10        else:
11            word_count[word] = 1
12    return word_count
13
14 text_dict = word_count(words)
15
16
17 for i in text_dict:
18     print(i, text_dict[i])
```

Listing 4: Dictionary Operations Implementation

2.4.3 Output

Hello 1
there, 1
I'm 1
Sabbir 1
Ahmed, 1
a 2
passionate 1
and 4
curious 1
individual 1
from 1
Dhaka, 1
Bangladesh. 1
As 1
Computer 1
Science 1
Engineering 1
student 1
at 1
Green 1
University 1
of 1
Bangladesh, 1
I 1
am 1
constantly 1
seeking 1
new 1
opportunities 1
to 1
expand 1
my 1
knowledge 1
skills 1
in 1
the 1
software 1
engineering 1
data 1
science 1
field. 1

```
Dictionary.py > word_count
1 text = "Hello there, I'm Sabbir Ahmed, a passionate and curious individual from Dhaka, Bangladesh. As a Computer Science Engineering student"
2
3 words = text.split()
4
5 def word_count(words):
6     word_count = {}
7     for word in words:
8         if word in word_count:
9             word_count[word] += 1
10        else:
11            word_count[word] = 1
12    return word_count
13
14 text_dict = word_count(words)
15
16 for i in text_dict:
17     print(i, text_dict[i])
18
19
20
```

```
PS C:\Users\msa29\Desktop\Python-Practice> & C:/Users/msa29/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Python-Practice/Dictionary.py
Hello 1
there, 1
I'm 1
Sabbir 1
Ahmed, 1
a 2
passionate 1
and 4
curious 1
individual 1
from 1
Dhaka, 1
Bangladesh. 1
As 1
Computer 1
Science 1
Engineering 1
student 1
```

Figure 4: Python Dictionary

3 NumPy Operations

3.1 Matrix Operations



3.1.1 Objective

Generation and manipulation of a 5x5 random integer matrix, demonstrating NumPy's capabilities for matrix operations.

3.1.2 Technical Details

The implementation uses:

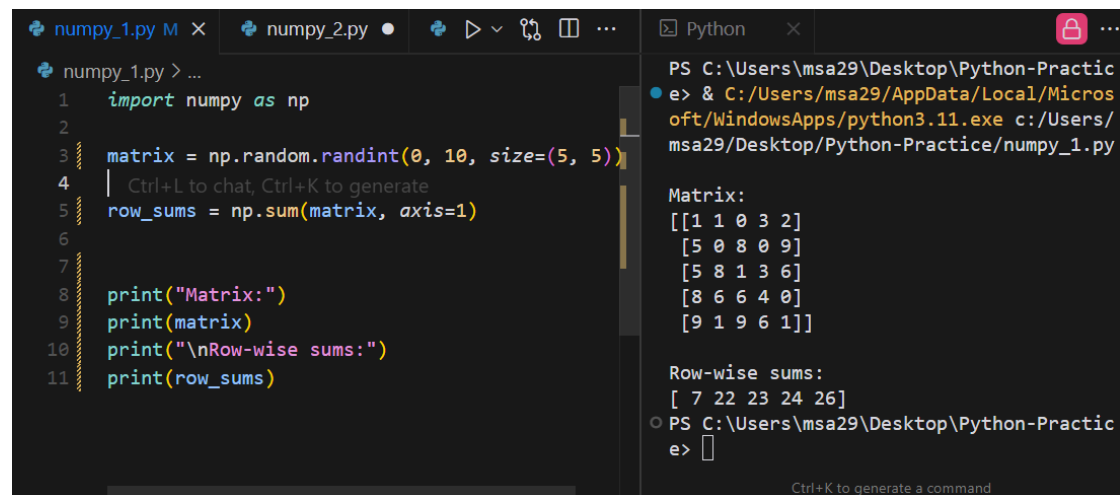
- `np.random.randint` for matrix generation
- `np.sum` with `axis` parameter for row-wise calculations

3.1.3 Implementation

```
1 import numpy as np
2
3 matrix = np.random.randint(0, 10, size=(5, 5))
4
5 row_sums = np.sum(matrix, axis=1)
6
7
8 print("Matrix:")
9 print(matrix)
10 print("\nRow-wise sums:")
11 print(row_sums)
```

Listing 5: Matrix Operations Implementation

3.1.4 Output



```
numpy_1.py M X numpy_2.py Python X
numpy_1.py > ...
1 import numpy as np
2
3 matrix = np.random.randint(0, 10, size=(5, 5))
4 | Ctrl+L to chat, Ctrl+K to generate
5 row_sums = np.sum(matrix, axis=1)
6
7
8 print("Matrix:")
9 print(matrix)
10 print("\nRow-wise sums:")
11 print(row_sums)
```

```
PS C:\Users\msa29\Desktop\Python-Practic
e> & C:/Users/msa29/AppData/Local/Micros
oft/WindowsApps/python3.11.exe c:/Users/
msa29/Desktop/Python-Practice/numpy_1.py

Matrix:
[[1 1 0 3 2]
 [5 0 8 0 9]
 [5 8 1 3 6]
 [8 6 6 4 0]
 [9 1 9 6 1]]

Row-wise sums:
[ 7 22 23 24 26]
PS C:\Users\msa29\Desktop\Python-Practic
e>
```

Figure 5: Numpy: row wise sum

3.2 Array Normalization



3.2.1 Objective

Generation and normalization of random arrays, demonstrating advanced array operations using NumPy.

3.2.2 Technical Details

The implementation includes:

- Generation of 100 random values using `np.random.rand`
- Normalization using the formula: $(x - \min(x)) / (\max(x) - \min(x))$

3.2.3 Implementation

```
1 import numpy as np
2
3 arr = np.random.rand(100)
4
5 print("Original array:")
6 print(arr)
7 print("\nOriginal array min:", np.min(arr))
8 print("Original array max:", np.max(arr))
9
10 normalized_arr = (arr - np.min(arr)) / (np.max(arr) - np.min(arr))
11
12 print("\nNormalized array:")
13 print(normalized_arr)
```



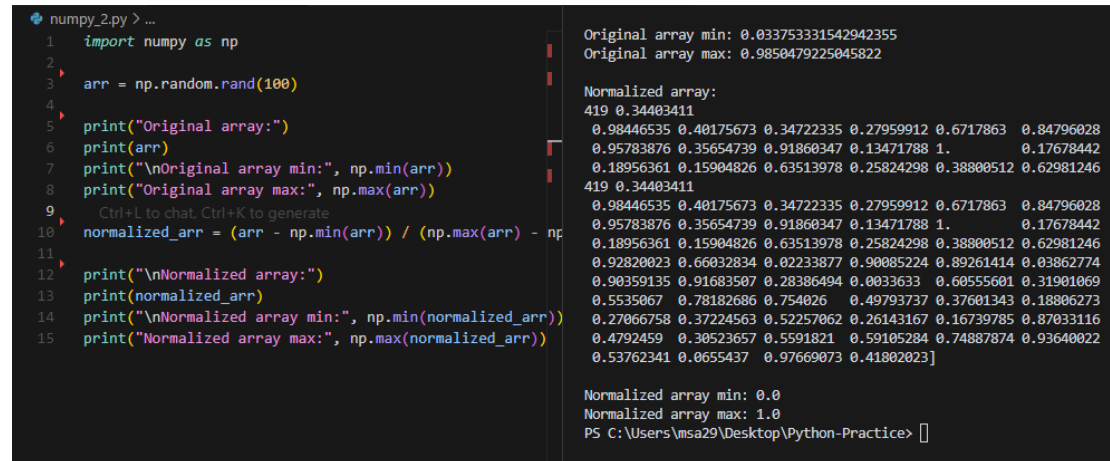
```

14 print("\nNormalized array min:", np.min(normalized_arr))
15 print("Normalized array max:", np.max(normalized_arr))

```

Listing 6: Array Normalization Implementation

3.2.4 Output



```

numpy_2.py > ...
1 import numpy as np
2
3 arr = np.random.rand(100)
4
5 print("Original array:")
6 print(arr)
7 print("\nOriginal array min:", np.min(arr))
8 print("Original array max:", np.max(arr))
9
10 normalized_arr = (arr - np.min(arr)) / (np.max(arr) - np.min(arr))
11
12 print("\nNormalized array:")
13 print(normalized_arr)
14 print("\nNormalized array min:", np.min(normalized_arr))
15 print("Normalized array max:", np.max(normalized_arr))

```

Original array min: 0.033753331542942355
Original array max: 0.9850479225045822

Normalized array:
419 0.34403411
0.98446535 0.40175673 0.34722335 0.27959912 0.6717863 0.84796028
0.95783876 0.35654739 0.91860347 0.13471788 1. 0.17678442
0.18956361 0.15904826 0.63513978 0.25824298 0.38800512 0.62981246
419 0.34403411
0.98446535 0.40175673 0.34722335 0.27959912 0.6717863 0.84796028
0.95783876 0.35654739 0.91860347 0.13471788 1. 0.17678442
0.18956361 0.15904826 0.63513978 0.25824298 0.38800512 0.62981246
0.92820023 0.66032834 0.02233877 0.90085224 0.89261414 0.03862774
0.90359135 0.91683507 0.28386494 0.0033633 0.60555601 0.31901069
0.5535067 0.78182686 0.754026 0.49793737 0.37601343 0.18806273
0.27066758 0.37224563 0.52257062 0.26143167 0.16739785 0.87033116
0.4792459 0.30523657 0.5591821 0.59105284 0.74887874 0.93640022
0.53762341 0.0655437 0.97669073 0.41802023]

Normalized array min: 0.0
Normalized array max: 1.0
PS C:\Users\msa29\Desktop\Python-Practice> █

Figure 6: Numpy Matrix Normalization

4 Pandas Data Analysis

4.1 Revenue Analysis



4.1.1 Objective

Analysis of furniture sales data using Pandas, demonstrating data grouping and aggregation techniques.

4.1.2 Technical Details

The implementation features:

- Data grouping by product category
- Revenue and sales calculations
- Advanced aggregation techniques

4.1.3 Implementation

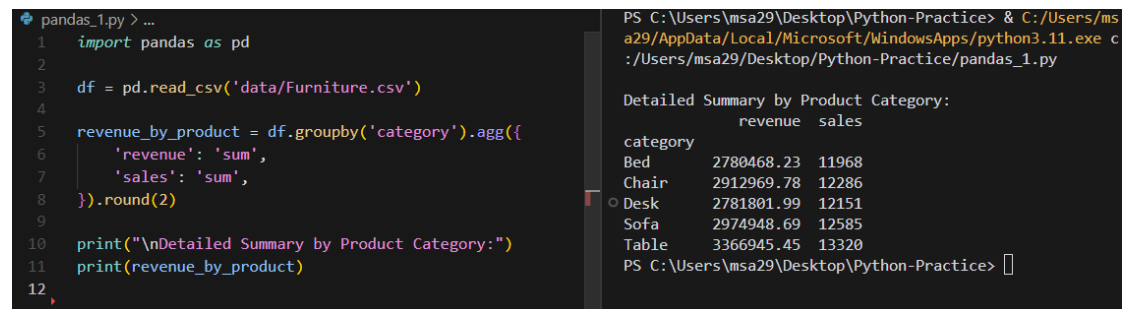
```

1 import pandas as pd
2
3 df = pd.read_csv('data/Furniture.csv')
4
5 revenue_by_product = df.groupby('category').agg({
6     'revenue': 'sum',
7     'sales': 'sum',
8 }).round(2)
9
10 print("\nDetailed Summary by Product Category:")
11 print(revenue_by_product)

```

Listing 7: Revenue Analysis Implementation

4.1.4 Output



```

PS C:\Users\msa29\Desktop\Python-Practice> & C:/Users/msa29/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Python-Practice/pandas_1.py

Detailed Summary by Product Category:
      revenue  sales
category
Bed      2780468.23  11968
Chair    2912969.78  12286
Desk     2781801.99  12151
Sofa     2974948.69  12585
Table    3366945.45  13320
PS C:\Users\msa29\Desktop\Python-Practice>

```

Figure 7: Sales Revenue

4.2 Missing Value Treatment



4.2.1 Objective

Handling missing values in furniture data, demonstrating data cleaning and preprocessing techniques.

4.2.2 Technical Details

Key features include:

- Null value detection using `isnull()`
- Numeric column separation
- Mean calculation and imputation

4.2.3 Implementation

```

1 import pandas as pd
2
3 df = pd.read_csv('data/Furniture copy.csv')
4
5 print("Null values before filling:")
6 print(df.isnull().sum())
7
8 numeric_cols = df.select_dtypes(include=['float64', 'int64']).
    columns
9 column_means = df[numeric_cols].mean()
10
11 df_filled = df.copy()
12 df_filled[numeric_cols] = df_filled[numeric_cols].fillna(
    column_means)
13
14 print("\nNull values after filling:")
15 print(df_filled.isnull().sum())
16
17 df_filled.to_csv('data/Furniture_filled.csv', index=False)
18
19 original_null_rows = df[df.isnull().any(axis=1)]
20 filled_null_rows = df_filled.loc[original_null_rows.index]
21 print("\nSample of filled rows (before and after):")
22 print("\nBefore filling:")
23 print(original_null_rows)
24 print("\nAfter filling:")
25 print(filled_null_rows)

```

Listing 8: Missing Value Treatment Implementation

4.2.4 Output

```

pandas_2.py > ...
1 import pandas as pd
2
3 # Read the CSV file
4 df = pd.read_csv('data/Furniture copy.csv')
5 # Print number of null values before filling
6 print("Null values before filling:")
7 print(df.isnull().sum())
8
9 # Calculate column-wise means for numeric columns only
10 numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
11 column_means = df[numeric_cols].mean()
12
13 # Fill missing values with column means
14 df_filled = df.copy()
15 df_filled[numeric_cols] = df_filled[numeric_cols].fillna(column_means)
16
17 # Print number of null values after filling
18 print("\nNull values after filling:")
19 print(df_filled.isnull().sum())
20

```

```

PS C:\Users\msa29\Desktop\Python-Practice> & C:/Users/msa29/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Python-Practice/pandas_2.py
Null values before filling:
price      1
cost       0
sales      2
profit_margin  0
inventory  2
discount_percentage  1
delivery_days  1
category   0
material   0
color      0
location   0
season     0
store_type  0
brand      0
revenue    0
dtype: int64

Null values after filling:

```

Figure 8: Missing Value Before Filling

```

pandas_2.py > ...
1 import pandas as pd
2
3 # Read the CSV file
4 df = pd.read_csv('data/Furniture copy.csv')
5 Ctrl+L to chat, Ctrl+K to generate
6 # Print number of null values before filling
7 print("Null values before filling:")
8 print(df.isnull().sum())
9
10 # Calculate column-wise means for numeric columns only
11 numeric_cols = df.select_dtypes(include=['float64', 'int64'])
12 column_means = df[numeric_cols].mean()
13
14 # Fill missing values with column means
15 df_filled = df.copy()
16 df_filled[numeric_cols] = df_filled[numeric_cols].fillna(c
17
18 # Print number of null values after filling
19 print("\nNull values after filling:")
20 print(df_filled.isnull().sum())

```

```

dtype: int64

Null values after filling:
price      0
cost       0
sales      0
profit_margin  0
inventory  0
discount_percentage  0
delivery_days  0
category   0
material   0
color      0
location   0
season     0
store_type  0
brand      0
revenue    0
dtype: int64

Sample of filled rows (before and after):

```

Ctrl+K to generate a command

Figure 9: Missing Value after Filling

5 Data Visualization with Matplotlib

5.1 Temperature Variation Analysis



5.1.1 Objective

Visualization of weekly temperature variations using line plots, demonstrating time series data visualization.

5.1.2 Technical Details

Implementation features:

- DateTime conversion and processing
- Daily temperature aggregation
- Custom plot styling and formatting

5.1.3 Implementation

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_csv('data/first_week_weather.csv')
5
6 data['Date_Time'] = pd.to_datetime(data['Date_Time'])
7
8 daily_temps = data.groupby(data['Date_Time'].dt.date)['Temperature_C'].mean().reset_index()
9
10 daily_temps = daily_temps.sort_values('Date_Time').head(7)

```

```

11
12 plt.figure(figsize=(10, 5))
13 plt.plot(daily_temps['Date_Time'], daily_temps['Temperature_C'],
14          marker='o', linestyle='--', color='b', linewidth=2,
15          markersize=8)
16
17 plt.title('Average Daily Temperature Variations Over a Week')
18 plt.xlabel('Date')
19 plt.ylabel('Temperature (\textdegree C)')
20 plt.xticks(rotation=45)
21 plt.grid(True)
22 plt.tight_layout()
23 plt.show()

```

Listing 9: Temperature Analysis Implementation

5.1.4 Output

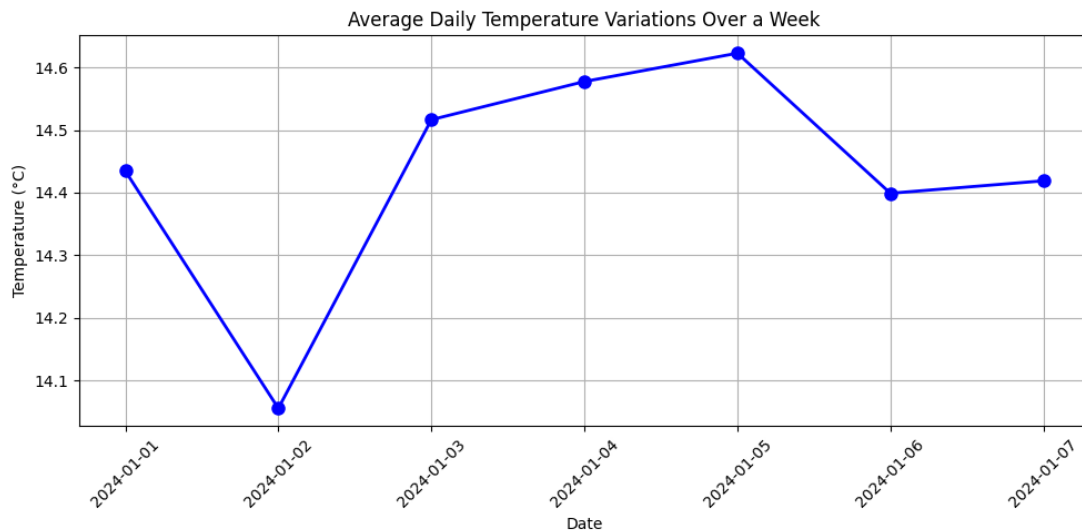


Figure 10: Line chart showing the average daily temperature variations over the first week of 2024. The plot demonstrates clear temperature fluctuations, with a notable peak on January 5th and a minimum on January 2nd. The visualization includes data points marked with blue circles and connected by lines for better trend visibility.

5.2 Regional Sales Analysis



5.2.1 Objective

Creation of bar charts for regional Mac sales analysis, demonstrating categorical data visualization.

5.2.2 Technical Details

Key features include:

- Bar chart creation with value labels
- Custom formatting and styling
- Grid lines and layout optimization

5.2.3 Implementation

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 data = pd.read_csv('data/apple_sales_2024.csv')
5
6 regional_sales = data.groupby('Region')['Mac Sales (in million
   units)'].mean()
7
8 plt.figure(figsize=(12, 6))
9 bars = plt.bar(regional_sales.index, regional_sales.values)
10
11 plt.title('Average Mac Sales by Region', fontsize=14, pad=20)
12 plt.xlabel('Region', fontsize=12)
13 plt.ylabel('Average Mac Sales (Million Units)', fontsize=12)
14
15 plt.ylim(5, 6)
16
17 plt.xticks(rotation=45)
18
19 for bar in bars:
20     height = bar.get_height()
21     plt.text(bar.get_x() + bar.get_width()/2., height,
22             f'{height:.2f}M',
23             ha='center', va='bottom')
24
25 plt.grid(axis='y', linestyle='--', alpha=0.7)
26
27 plt.tight_layout()
28
29 plt.show()

```

Listing 10: Regional Sales Analysis Implementation

5.2.4 Output

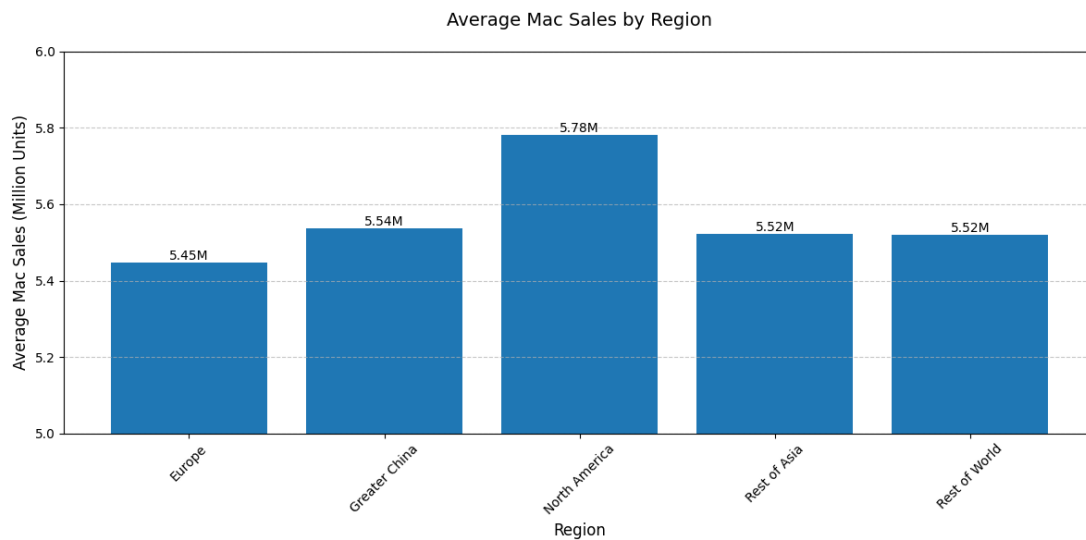


Figure 11: Bar chart comparing average Mac sales across different regions. North America shows the highest sales at 5.78M units, followed by Greater China at 5.54M units. The visualization includes value labels on top of each bar and a consistent grid pattern for easy comparison.

6 Conclusion

This report demonstrates the implementation of various Python programming concepts, from basic data structures to advanced data analysis and visualization techniques. The implementations showcase the versatility of Python and its libraries in handling different types of data processing and analysis tasks.

7 References

1. NumPy Documentation - <https://numpy.org/doc/>
2. Pandas Documentation - <https://pandas.pydata.org/docs/>
3. Matplotlib Documentation - <https://matplotlib.org/stable/contents.html>
4. Python Documentation - <https://docs.python.org/3/>
5. GeeksforGeeks - Python Lists - <https://www.geeksforgeeks.org/python-ways-to-remove-d>
6. W3Schools - Python Sets - https://www.w3schools.com/python/ref_set_intersection.asp