

Solutions

6.1 Arrays and Vectorised Operations

Reference example

```
import numpy as np
a = np.array([1, 2, 3])
print((a * 2).tolist())
print((a + 10).tolist())
```

6.1.1 - Trace

Determine the exact output and explain the state changes.

```
import numpy as np
a = np.array([2, 4, 6])
print(a[a > 3].tolist())
```

Output

[4, 6]

The Boolean mask selects the entries whose values exceed 3.

6.1.2 - Write code

Use a NumPy array to square [2, 3, 4] elementwise and print an ordinary list.

Exponentiation is elementwise for this array.

```
import numpy as np
a = np.array([2, 3, 4])
print((a ** 2).tolist())
```

Expected output

[4, 9, 16]

Solutions

6.2 DataFrames

Reference example

```
import pandas as pd
df = pd.DataFrame({"name": ["Ada", "Lin"], "score": [7, 9]})
print(df["score"].sum())
print(df.loc[df["score"] >= 8, "name"].tolist())
```

6.2.1 - Trace

Determine the exact output and explain the state changes.

```
import pandas as pd
df = pd.DataFrame({"x": [2, 5]})
df["double"] = 2 * df["x"]
print(df["double"].tolist())
```

Output

[4, 10]

The new column is calculated from each corresponding x value.

6.2.2 - Write code

Create a DataFrame with scores [4, 8, 6] and print scores at least 6.

Filter with a Boolean mask.

```
import pandas as pd
df = pd.DataFrame({"score": [4, 8, 6]})
print(df.loc[df["score"] >= 6, "score"].tolist())
```

Expected output

[8, 6]

Solutions

6.3 Cleaning and Missing Data

Reference example

```
import pandas as pd
scores = pd.Series([5.0, None, 9.0])
print(int(scores.isna().sum()))
print(scores.dropna().mean())
```

6.3.1 - Explain

How would replacing the missing value with zero change the mean of 5, missing, 9?

The observed-only mean is 7. Replacing missing with zero gives 14/3, about 4.667, asserting an unobserved zero and changing the interpretation.

6.3.2 - Write code

Count missing entries in [1.0, None, None, 4.0] using pandas.

Use an explicit missingness check.

```
import pandas as pd
values = pd.Series([1.0, None, None, 4.0])
print(int(values.isna().sum()))
```

Expected output

2

Solutions

6.4 Grouping and Summarising

Reference example

```
import pandas as pd
df = pd.DataFrame({"team": ["A", "B", "A"], "score": [4, 9, 8]})
means = df.groupby("team")["score"].mean()
for team in sorted(means.index):
    print(team, float(means[team]))
```

6.4.1 - Explain

Why can averaging group averages give the wrong overall mean when group sizes differ?

It gives equal weight to groups rather than observations. Use counts and sums, or an appropriate weighted mean, when calculating the overall observation mean.

6.4.2 - Write code

For groups A, A, B with values 2, 6, 5, print each group total in sorted order.

Group by the category and sum the value column.

```
import pandas as pd
df = pd.DataFrame({"group": ["A", "A", "B"], "value": [2, 6, 5]})
totals = df.groupby("group")["value"].sum()
for group in sorted(totals.index):
    print(group, int(totals[group]))
```

Expected output

```
A 8
B 5
```