Vectorized Operations





intro

Vectorized operations allow you to perform operations on entire arrays or Series (collections of data) without using explicit loops.

- Why It's Important:
- •Efficiency: Faster execution than loops.
- Cleaner Code: More readable and concise.
- Parallelism: Many vectorized operations are optimized to run on multiple CPU cores, making them faster for large datasets.

Examples of vectorized operations in Pandas:

- where()
- select()



Why Vectorized Operations Matter

•Performance:

- Vectorized operations are implemented in C (underlying NumPy and pandas), making them much faster than Python loops.
- Example: Adding two large lists using loops vs. vectorized operations.

•Memory Efficiency:

Operations on entire arrays are performed in a single pass, avoiding the overhead of Python loops.



Vectorized vs. loop

[5, 7, 9]

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = []
for i in range(len(a)):
    result.append(a[i] + b[i])
result
```

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = a + b
print(result)
[5 7 9]
```

- •Loops: Slower because Python executes the loop one iteration at a time.
- •Vectorized: Much faster, executed in C (internally optimized).



The apply Method -

Purpose: Apply a function to each element, row, or column of a DataFrame or Series.

```
Series.apply(func)
DataFrame.apply(func, axis=0)

Lambda or predefined function # axis=0 for columns, axis=1 for rows
```

Code example

```
df.apply(lambda row: row['A'] + row['B'], axis=1)
```



The apply Method -

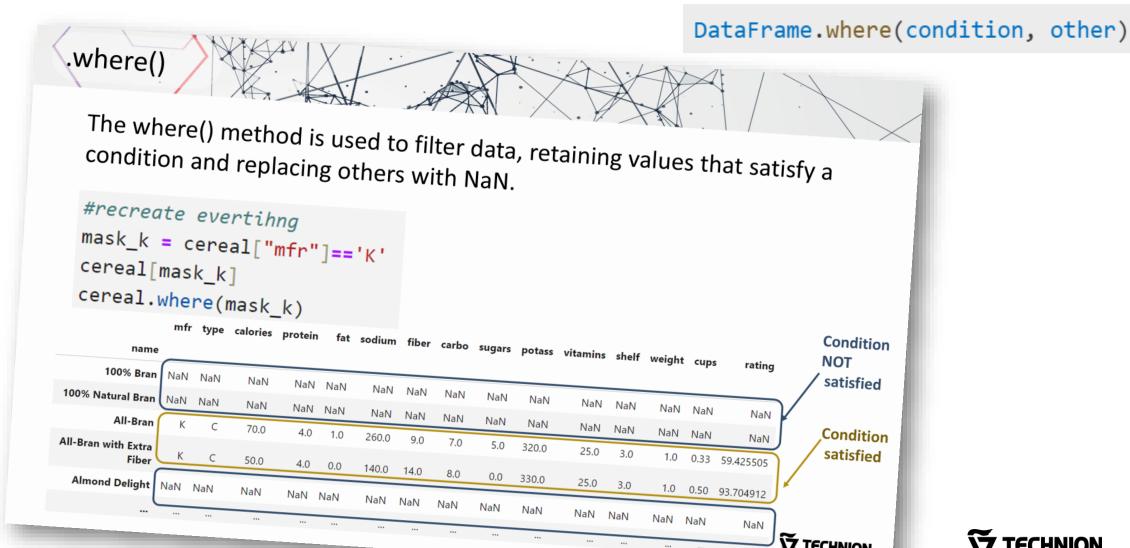
apply is **not truly vectorized**.

- While it is more concise and easier to use than explicit loops, it still operates element-by-element or row-by-row.
- Internally, apply often calls Python functions (like lambda), which makes it slower than fully vectorized operations.
- For large datasets, the overhead of Python function calls in apply adds up and can make it significantly slower than NumPy-based or vectorized operations.



The where Method

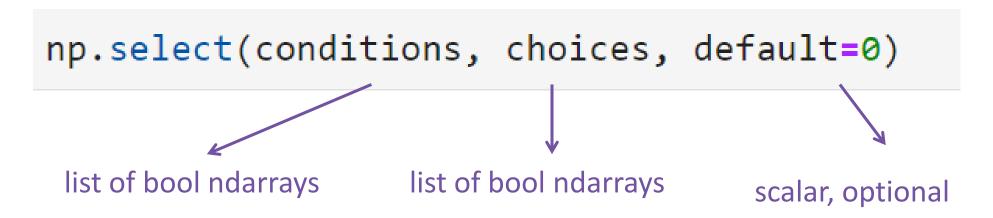
Purpose: Conditionally replace values in a DataFrame or Series





The select Function

Purpose: Conditionally replace values in a DataFrame or Series.



```
conditions = [df['Fare']> 50, df['Fare']<20]
outcomes = ['Expensive', 'Cheep']
np.select(conditions,outcomes, 'ok')</pre>
```



The select Function

```
conditions = [df['Fare'] > 50, df['Fare'] < 20]
outcomes = ['Expensive', 'Cheep']
np.select(conditions,outcomes, 'ok')</pre>
select se example
```

Concat the "select" column to the data frame

to the data i	0	Fare	Age	Sex	Name	
			7.90			
	Sir	7.2500	22.0	male	Mr. Owen Harris Braund	0
	adult	71.2833	38.0	female	Mrs. John Bradley (Florence Briggs Thayer) Cum	1
Final result	She/Her	7.9250	26.0	female	Miss. Laina Heikkinen	2
	adult	53.1000	35.0	female	Mrs. Jacques Heath (Lily May Peel) Futrelle	3
	Sir	8.0500	35.0	male	Mr. William Henry Allen	4
TEC Azrieli Cont External St					•••	



Best Practices

- Use vectorized operations whenever possible.
- Avoid explicit loops over DataFrame rows or columns.
- Combine apply, where, and select for advanced use cases.
- Leverage NumPy functions when required for additional

functionality.



Feature	apply	where	np.select
Vectorized?	No	Yes	Yes
Flexibility	Very high (supports custom functions)	Medium (conditionally replaces values)	High (handles multiple conditions)
Performance	Slower (element- wise operations)	Faster (column/array-based)	Faster (multiple vectorized conditions)
Use Case	Complex logic with custom functions	Simple conditional replacements	Multiple complex conditions



- Vectorized operations streamline data analysis in Pandas.
- Key functions discussed:
 - apply for applying functions element-wise
 - where for conditional updates
 - select for multi-condition choices



Additional Resources

- Pandas Documentation: https://pandas.pydata.org/docs/
- •NumPy Documentation: https://numpy.org/doc/stable/

