

# Week 1: Intro & Data Lifecycle

## Data Analytics Overview

- **Definition:** Process of examining data sets to draw conclusions about the information they contain.
- **Goal:** Extract actionable insights, support decision-making.

## Data Lifecycle Stages

1. **Generation:** Data creation (IoT sensors, user input, logs).
2. **Collection:** Gathering raw data (APIs, web scraping, surveys).
3. **Processing:** Cleaning, formatting, normalizing (ETL).
4. **Storage:** DBs (SQL/NoSQL), Data Lakes, Warehouses.
5. **Management:** Governance, security, access control.
6. **Analysis:** EDA, statistical modeling, ML.
7. **Visualization:** Charts, dashboards for communication.
8. **Interpretation:** Deriving business value/insights.

## Week 2: NumPy Basics

Import: import numpy as np

### Array Creation

- `np.array([1, 2, 3])`: Create 1D array.
- `np.zeros((2, 3))`: 2x3 array of zeros.
- `np.ones((2, 3))`: 2x3 array of ones.
- `np.arange(0, 10, 2)`: [0, 2, 4, 6, 8].
- `np.linspace(0, 1, 5)`: 5 evenly spaced values from 0 to 1.
- `np.random.rand(2, 2)`: Random values [0, 1).

### Attributes

- `arr.ndim`: Number of dimensions.
- `arr.shape`: Tuple of dimensions (e.g., (2, 3)).
- `arr.size`: Total number of elements.
- `arr.dtype`: Data type of elements (int, float, etc.).

### Reshaping & Flattening

- `arr.reshape((new_r, new_c))`: Returns new shape view (no data copy).
- `arr.flatten()`: Returns **copy** 1D array.
- `arr.ravel()`: Returns **view** 1D array (modifies original if changed).
- `arr.transpose()` or `arr.T`: Transpose array.

### Broadcasting

Automatic expansion of arrays for arithmetic operations.

- Rules: Compare dimensions from right to left.
- Compatible if: Dimensions are equal OR one is 1.
- Example:  $A(3, 1) + B(1, 3) \rightarrow \text{Result}(3, 3)$ .
- `X[:, np.newaxis]`: Adds dimension for broadcasting.

## Week 3: Adv. NumPy & K-NN

### File I/O

- `np.loadtxt('file.csv', delimiter=',')`: Load simple text data.
- `np.genfromtxt`: Handles missing values better.
- `np.save('file.npy', arr) / np.load()`: Binary storage.

### Indexing & Selection

- `arr[0, 1]`: Row 0, Col 1.
- `arr[0:2, :]`: Slicing rows 0 to 1, all cols.
- `arr[arr > 5]`: Boolean mask filtering.
- `np.where(cond, x, y)`: If cond True x, else y.

## Advanced Math Ops

- `np.sum(arr, axis=0)`: Sum cols (downwards).
- `np.sum(arr, axis=1)`: Sum rows (across).
- `np.mean()`, `np.std()`, `np.var()`, `np.min()`, `np.max()`.
- `np.matmul(A, B)` or `A @ B`: Matrix multiplication.
- `np.dot(a, b)`: Dot product.

## Nearest Neighbor (K-NN) Logic

Distance Calculation (Euclidean):

- Squared Dist:  $\|x - y\|^2 = \sum (x_i - y_i)^2$ .
- Vectorized pairwise distance (Assignment 2 logic):
  - $D^2 = \|A\|^2 + \|B\|^2 - 2A \cdot B^T$ .
  - `sq_norms = np.sum(X**2, axis=1)`
  - `dist = sq_norms[:, None] + sq_norms[None, :] - 2 * X @ X.T`
- `np.argsort(dist, axis=1)`: Get indices of sorted neighbors.
- `indices[:, 1:k+1]`: Exclude self (index 0), get k neighbors.

## Week 4: Pandas Intro

Import: import pandas as pd

### Pandas Objects

- **Series**: 1D array with index.
- `pd.Series([10, 20], index=['a', 'b'])`
- **DataFrame**: 2D tabular data with row/col labels.
- `pd.DataFrame(dict)`: From dictionary.
- `pd.DataFrame(arr, columns=['A', 'B'])`: From NumPy.

### Series Attributes

- `s.values`: Returns numpy array of data.
- `s.index`: Returns index object.
- `s['a']`: Access by index label.

### DataFrame Basics

- `df.head(n) / df.tail(n)`: First/last n rows.
- `df.info()`: Types, non-null counts.
- `df.describe()`: Summary stats (mean, std, min, max).
- `df.shape`, `df.columns`, `df.index`.

## Week 5: Pandas Indexing

### Selection Methods

- `.loc[]`: Label-based selection.
- `df.loc['row_label', 'col_label']`
- `df.loc['r1':'r3', ['c1', 'c2']]` (Inclusive end).
- `.iloc[]`: Integer position-based.
- `df.iloc[0, 1]`: Row 0, Col 1.
- `df.iloc[0:3, 0:2]` (Exclusive end, standard Python).
- `df['col']`: Select column as Series.
- `df[['c1', 'c2']]`: Select cols as DataFrame.

### Boolean Masking

- `df[df['A'] > 5]`: Filter rows where col A > 5.
- `mask = (df['A'] > 0) & (df['B'] < 10)`: Multiple conditions (use `&`, `|`).
- `df.loc[mask, 'C']`: Filter rows, select col C.

### Data Modification

- `df['new_col'] = val`: Add/Update column.
- `df.drop('col', axis=1)`: Drop column.
- `df.drop('index', axis=0)`: Drop row.
- `df.rename(columns={'old': 'new'})`: Rename.

## Sorting

- `df.sort_values(by='col')`: Sort by column values.
- `df.sort_index()`: Sort by index labels.
- `ascending=False`: For descending order.

## Assign 1: Floating Points

**Key Concept:** Binary floating point is inexact ( $0.1 + 0.2 \neq 0.3$ ).

### Robust Comparison

- **Do NOT use:** `a == b` (fails for floats/NaNs).
- **Use:** `np.isclose(a, b, atol=1e-9)`.

### NaN Handling

- Standard: `np.nan == np.nan` is **False**.
- Robust: `np.isclose(..., equal_nan=True)`.
- Pandas: `df.fillna()`, `df.dropna()`.

## Assign L1: Importing Data

### Core Methods

1. **CSV:** `pd.read_csv('file.csv')`.
2. **ZIP:** Use `zipfile` module.
  - `with zipfile.ZipFile('f.zip') as z: z.extractall()`
3. **SQL:** `sqlite3` + Pandas.
  - `conn = sqlite3.connect('db.sqlite')`
  - `pd.read_sql_query("SELECT * FROM T", conn)`
4. **JSON:** Nested data.
  - `import json; data = json.load(open('f.json'))`
  - Drill down dicts before `pd.DataFrame(data['key'])`.
5. **Stata:** `pd.read_stata('f.dta')`.
  - Use `reader.variable_labels()` for metadata.

## Assign 2: Greedy Heuristic

**Problem:** Select subset to label such that every unlabeled point has a labeled neighbor.

### Algorithm Steps

1. **Pairwise Distance:** Compute  $(N, N)$  matrix using broadcasting (No loops!).
2. **Neighbors:** `np.argsort` to get nearest 3 indices.
3. **Init:** Randomly label 5 points.
4. **Loop:** Find unlabeled points with NO labeled neighbors.
5. **Greedy Choice:** For violations, label the **closest** neighbor.
6. **Verify:** Check all 0s have a 1 neighbor.

### Vectorization Tips

- Avoid **for** loops over arrays!
- Use `np.where` for conditional logic.
- Use broadcasting for element-wise ops across dimensions.

## Common Errors / Tips

- `axis=0`: Direction along rows (vertical operation).
- `axis=1`: Direction along cols (horizontal operation).
- `df.iloc` slicing is exclusive [start:end].
- `df.loc` slicing is inclusive [start:end].
- Parentheses needed for multiple boolean conditions: `(df['a']>1) & (df['b']<2)`.
- `ravel()` returns a view; modifying it modifies original.
- `flatten()` returns a copy.

## Chapter 2: NumPy Deep Dive

### 02.01: Data Types & Creation (Definitions)

- **Fixed-Type Arrays:** Unlike Python lists, NumPy arrays have a single dtype. This allows for memory-efficient storage and fast operations.
- `np.empty((2,2))`: Allocates memory without initializing values (faster, but contains "garbage" data).
- `np.eye(3)`:  $3 \times 3$  Identity matrix (1s on diagonal).
- `np.linspace(start, stop, num)`: Generates num samples over an interval. End is **included** by default.
- `np.random.normal(mu, sigma, shape)`: Draws random samples from a Gaussian distribution.

### 02.02: Basics (Attributes, Slicing, Views)

- **.itemsize**: Size of one element in bytes. **.nbytes**: Total bytes (`size * itemsize`).
- **Slicing Syntax**: `x[start:stop:step]`. Negative step (e.g., `::-1`) reverses the array.
- **Multi-dim Slicing**: `x[:,2, :3]` (rows 0-1, cols 0-2).
- **CRITICAL: Views vs. Copies**: Slicing returns a **view**. Modifying `sub = x[:2]` modifies `x`. Use `.copy()` to force a copy.
- **Reshaping**: `x.reshape((1, N))`  $\neq$  `x.reshape(N, 1)`.
- **New Axis**: `x[np.newaxis, :]` creates a row vector; `x[:, np.newaxis]` creates a column vector.

### 02.03: UFuncs (Universal Functions)

- **Vectorization**: Performing an operation on an entire array instead of looping. Uses C-optimized loops.
- `np.add`, `np.subtract`, `np.multiply`, `np.exp`, `np.log`, `np.power`.
- **Out Argument**: `np.multiply(x, 10, out=y)` writes result directly to `y` (memory efficient).

### 02.04: Aggregations (Axis Logic)

- **Reduction**: Collapsing dimensions.
- **Axis 0 (Columns)**: Operation works "down" the rows. Result has same number of columns as input.
- **Axis 1 (Rows)**: Operation works "across" the columns. Result has same number of rows as input.
- `np.nansum()`, `np.nanmean()`: Safely ignore NaN values. Standard `np.sum()` returns NaN if any element is NaN.

### 02.05: Broadcasting (The 3 Rules)

- **Rule 1**: If arrays have different ndim, the shape of the one with fewer dimensions is **padded with 1s on its left side**.
- **Rule 2**: If any dimension size is 1, that dimension is stretched to match the other array.
- **Rule 3**: If in any dimension sizes mismatch and neither is 1, an error is raised.

### 02.06: Comparisons & Masks

- **Comparison Operators**: `<`, `>`, `<=`, `>=`, `==`, `!=` return a boolean array.
- **Bitwise Logic**: Use `&` (and), `|` (or), `~` (not). Standard Python and/or do not work on arrays.
- **Parens**: Always wrap conditions in parentheses: `(x > 1) & (x < 5)`.

### 02.07: Fancy Indexing

- Passing an array of indices to access multiple elements.

- **Important**: The shape of the result reflects the shape of the **index arrays**, not the original array.
- Modifying values: `x[indices] = val` updates multiple spots at once.

### 02.08: Sorting & Partitioning

- `np.sort()`: Returns sorted copy. `x.sort()`: Sorts in-place.
- `np.argsort()`: Returns the **indices** that would sort the data. Essential for K-NN.
- `np.partition(x, K)`: Places the *K* smallest values on the left (unsorted) and the rest on the right. Faster than full sort if you only need the top *K*.

## Chapter 3: Pandas Indexing

### 03.01: Introducing Pandas Objects

- **Series**: A 1D array with an explicit index. Behaves like a hybrid between a NumPy array and a Python dictionary.
- **DataFrame**: A 2D table. Can be viewed as a sequence of aligned Series objects sharing the same index.
- **Index Object**: An immutable array and an ordered set. Allows for label-based alignment.

### 03.02: Indexing and Selection (Exam Prep)

- **Implicit vs. Explicit**: `s[0]` might refer to the first element (implicit) or the label 0 (explicit).
- **.loc (Label-based)**: Always refers to the explicit label. Slicing `['a': 'c']` is **INCLUSIVE** of 'c'.
- **.iloc (Integer-based)**: Always refers to the implicit position (0, 1, 2...). Slicing `[0:2]` is **EXCLUSIVE** of 2 (returns 0, 1).
- **Masking**: `df[df['A'] > 5]` works identically to NumPy.

## Assignment Core Math & Logic

### Assign 1: Floating Point Ops

- **The IEEE 754 Trap**: `0.1 + 0.2 != 0.3` due to binary rounding.
- **The NaN Trap**: `np.nan == np.nan` is **False**.
- **Solution**: `np.isclose(a, b, atol=1e-9, equal_nan=True)`.
- `equal_nan=True` is mandatory for treat NaNs as a match.

### L1: Importing Data Types

- **ZIP**: `zipfile.ZipFile(path).extractall(folder)`.
- **SQL**: `sqlite3.connect(db_path) → pd.read_sql_query(query, conn)`.
- **JSON Drill-down**: `json.load(f)` returns a dict. Access nested levels via `dict['level1']['level2']` before passing to `pd.DataFrame()`.
- **Stata Metadata**: `pd.io.stata.StataReader(path).variable_labels()`.

### Assign 2: Greedy NNL Logic

- **Distance Matrix (No Loops)**:  
$$||x - y||^2 = \sum x^2 + \sum y^2 - 2xy.$$
- `sq = np.sum(X**2, axis=1)`
- `D2 = sq[:, np.newaxis] + sq[np.newaxis, :] - 2 * X @ X.T`
- **Neighbor Selection**: `np.argsort(D2, axis=1)[:, 1:4]` (skip index 0, self).
- **Greedy Logic**: Check `needs_label == 0` rows. If neighbor sum is 0, set `needs_label = 1` for the **closest** neighbor.

## Quick Exam Reference

### Common Syntax Errors

- **Axis**: `axis=0` (Col stats/Vertical), `axis=1` (Row stats/Horizontal).
- **Drop**: `df.drop('col', axis=1)` (Must specify `axis=1`).
- **Shapes**: `X.shape` is a tuple. `X.shape[0]` is rows, `X.shape[1]` is cols.
- **Deprecation**: `.ix` is removed. Use only `.loc` or `.iloc`.

### Codebook: Function Comparisons

Task	NumPy	Pandas
Index	<code>arr[0]</code>	<code>df.iloc[0]</code>
Labels	N/A	<code>df.loc['a']</code>
Logic	<code>&amp;</code> , <code> </code> , <code>~</code>	<code>&amp;</code> , <code> </code> , <code>~</code>
Sort	<code>np.sort()</code>	<code>df.sort_values()</code>
Indices	<code>np.argsort()</code>	<code>df.index</code>
Nulls	<code>np.isnan()</code>	<code>df.isnull()</code>