DS UNIT 2

Created	@May 13, 2025 12:53 PM

Unit 2: Data Wrangling, Data Cleaning and Preparation

1. Data Handling: Challenges with Large Data

Challenges in Handling Large Volumes of Data

- **Memory Limitations:** RAM may not fit the entire dataset. Swapping to disk slows processing.
- **Algorithm Scalability:** Many algorithms expect the whole dataset in memory, leading to "out-of-memory" errors.
- **Performance Issues:** Large data causes slow read/write (I/O) and processing (CPU).

Main Problems

- Out-of-memory errors.
- Algorithms running indefinitely (never-ending computations).
- I/O and CPU starvation (processes waiting on slow hardware resources).

---

2. Techniques for Handling Large Data Sets

a. Algorithm Design Approaches

- **Online Learning:** Data processed one sample at a time.
- **Mini-batch Learning:** Process small subsets (batches) instead of full data.
- **Batch Learning:** Classic approach, requires all data at once (impractical for big data).

b. Divide and Conquer Techniques

- **Split large matrices:** Use libraries (e.g., bcolz) to chunk data into manageable pieces.
- **Parallel & Distributed Computing:** (e.g., Dask, MapReduce).
  - **MapReduce:** Distribute tasks (e.g., sums/counts) across machines, then aggregate.

c. Use the Right Data Structures

- **Sparse Matrices:** Efficient for datasets with many zeros.
- **Trees:** For fast lookup, indexing, and retrieval (e.g., B-trees in databases).
- **Hash Tables:** Quick retrieval using hash keys (Python's dict is a hash table).

d. Choose Appropriate Tools & Libraries

- **bcolz:** Compressed arrays, operates out-of-core (beyond RAM).
- **Dask:** Parallel, scalable computations; can replace Pandas for big data.
- **MapReduce:** For parallel/distributed operations.
- **Cython:** Compile Python code for speed.
- **Numexpr:** Fast, multi-threaded computation on NumPy arrays.
- **Numba:** Just-in-time (JIT) compiling for Python.
- **Blaze:** Interface for working with out-of-core and database-backed arrays.
- **Theano:** Fast computation for machine learning; uses GPU.

e. General Programming Tips

- Re-use existing, optimized libraries; don't reinvent the wheel.
- Get more from hardware: use compression, multithreading, and GPUs if possible.
- Minimize computing needs:
  - Use code profilers to identify bottlenecks.

- Use compiled extensions (C/C++, Fortran) for critical code.
- Use generators to avoid holding entire intermediate data in memory.
- Process by chunks (stream data), use sampling if needed.
- Apply mathematical simplifications where possible.

- Utilize databases for data manipulation and querying, leveraging built-in query optimizations (indices, views).

---

3. Data Wrangling

Definition and Objectives

- Also called **Data Munging**: Process of transforming raw data into a usable format.
- Assures quality, consistency, and value for analysis and modeling.
- Involves **discovering, structuring, cleaning, enriching, validating, and publishing data**.

Steps in Data Wrangling

1. **Discovery:** Understand and explore what's in the data.
2. **Organization:** Structure the dataset sensibly.
3. **Cleaning:** Remove or correct errors, outliers, missing values, duplicates.
4. **Enrichment:** Add or derive new features if required.
5. **Validation:** Apply rules and checks for consistency.
6. **Publishing:** Document and make accessible for analysis/use.

Use Cases

- **Fraud Detection:** Identifying unusual patterns in transactions, emails, chats, etc.
- **Customer Behaviour Analysis:** Deriving insights swiftly for business decisions.

Common Data Wrangling Tools

- **Excel/Power Query:** Manual, small-scale wrangling.
- **OpenRefine:** Automated, for cleaning and transformation (needs programming).
- **Tabula:** Extracts tables from PDFs.
- **Google DataPrep:** Explores and prepares data at scale.
- **Data Wrangler:** Interactive tool for cleaning, especially tabular data.
- **Plotly, Pandas (Python):** Data analysis, visualization, and wrangling.

Benefits

- Enhanced Data Consistency.
- Improved Analytical Insights.
- Cost- and Resource-Efficient model and decision-making.

---

4. Data Cleaning and Preparation

Cleaning, Transforming, and Merging Data (Pandas)

1. Cleaning

- **Handle Missing Values**
- **Remove Duplicates**
- **Resolve Inconsistencies** (e.g., capitalization)**Example:**

```
import pandas as pd
df = pd.DataFrame({'Name': ['Alice', None], 'Age': [25, None]})
df = df.dropna()   # Remove rows with missing data
df = df.fillna({'Name': 'Unknown', 'Age': df['Age'].mean()})
```

2. Transformation

- Change data types: `df['col'] = df['col'].astype(int)`
- Apply functions: `df['new'] = df['old'].apply(func)`
- Renaming columns: `df.rename(columns={'old': 'new'})`
- Create new columns or features.

---

3. Merging and Combining Data

- **merge()**: SQL-like joins on keys or columns.
- **concat()**: Stack datasets vertically or horizontally (`axis=0` or `1`).
- **join()**: Combines on indices or keys.
- **combine_first()**: Fill missing values in one DataFrame with another.**Example:**

```
# Left join on 'ID'
merged = pd.merge(df1, df2, on='ID', how='left')
# Combine with overlap
df1.combine_first(df2)
```

- Use the right method:
  - `concat()` for stacking.
  - `merge()` for joining on columns.
  - `combine_first()` for filling missing data.

---

5. Reshaping and Pivoting Data

Reshaping Operations

- **Pivoting (`pivot()` , `pivot_table()`):**
  - Rearranges (reshapes) data from long to wide format (unique values become columns).
  - Example:

```
df.pivot(index='Date', columns='City', values='Sales')
```

- **Melting (`melt()`):**
  - Converts wide data back into long format.
  - Example:

```
df.melt(id_vars='Date', var_name='City', value_name='Sales')
```

- **Stack/Unstack**
  - `stack()` turns columns into hierarchical rows (MultiIndex).
  - `unstack()` reverses the operation.
  - Used for converting between wide and long forms.

---

6. Handling Missing Data

Definition

- Missing values (NaN, blank, None, 'NA') are common in datasets, and must be handled carefully.

Why Handle Missing Data?

- To ensure data quality, unbiased results, effective model building, and correct interpretations.

Techniques

1. **Deletion Methods**:
   - **Listwise Deletion:** Drop entire rows/columns with missing data.
   - **Pairwise Deletion:** Use only available (complete) pairs for calculations.
2. **Imputation Methods**
   - **Mean/Median/Mode Imputation:** Replace missing values with mean, median, or mode.

- **K-Nearest Neighbors Imputation:** Fill missing values using similar (neighbor) records.
- **Regression Imputation:** Predict the missing value using a regression model.
- **Multiple Imputation:** Create multiple imputed datasets and combine results.

3. **Forward/Backward Fill:**
    - Carry previous/next value forward/back to replace NaNs.**Checking and Handling Example:**

```
df.isnull()          # Show missing data mask
df.isnull().sum()    # Number of missing values per column
df.dropna()          # Drop rows with any NaN
df.fillna(0)         # Replace NaN by 0
df['col'].fillna(df['col'].mean()) # Replace with mean
```

Choosing Approach:

- Depends on data amount, nature of missingness, and importance to analysis.

---

7. Data Transformation Techniques

Smoothing

- Remove noise/outliers (e.g., via moving average, binning, or smoothing functions).

Attribute Construction

- Create new features from existing ones for easier analysis.

Data Generalization

- Convert specific data into broader categories (e.g., age to age groups).
    - **Attribute:** More general value replaces specific (Age → Age group)

- **Hierarchy:** Use hierarchical levels (Make/Model/Type)
- **Numeric:** Convert continuous variables into ranges
- **Text:** Replace specific words with general ones

Data Aggregation

- Combine and summarize data (e.g., group by and aggregate sums, averages).

Data Discretization (Binning)

- Transform continuous variables into discrete bins:
  - **Equal-width:** Bins all have same range.
  - **Equal-frequency (quantile):** Same number of samples per bin.
  - **Custom:** User-defined bins.

```
import pandas as pd
df['binned'] = pd.cut(df['Column'], bins=3)
```

Normalization and Standardization

- **Normalization:** Scale features to range [0, 1]
  - Formula: `(x - min(x)) / (max(x) - min(x))`
- **Standardization:** Mean = 0, Std Dev = 1
  - Formula: `(x - mean) / std_dev`
- Important for ML algorithms relying on distance or gradients.

---

8. String Manipulation in Data Cleaning

Basic Operations

- Length: `len(text)`
- Access: `text[0]`, `text[-1]`
- Slicing: `text[0:5]`

- Reversing: `text[::-1]`

Case Conversion

- `.upper()`, `.lower()`, `.title()`, `.capitalize()`, `.swapcase()`

Concatenation and Repetition

- `str1 + str2`
- `str1 * 3`

Searching and Replacing

- `"sub" in text`, `text.find("sub")`, `text.replace("old", "new")`

Splitting and Joining

- `text.split(",")`
- `",".join([list])`

Removing Whitespace

- `text.strip()`, `.lstrip()`, `.rstrip()`

Formatting Strings

- f-strings: `f"My name is {name}"`
- `.format()`: `"My name is {}".format(name)`
- `%` operator: `"My name is %s" % name`

String Properties

- `.isalpha()`, `.isdigit()`, `.isalnum()`, `.islower()`, `.isupper()`, `.isspace()`

Reversing words in a sentence

- `' '.join(text.split()[::-1])`

---

9. Summarizing Data

Centrality

- **Mean:** Average
- **Median:** Middle value
- **Mode:** Most frequent value

Dispersion

- **Standard Deviation:** How far values spread from mean
- **Variance:** Average squared difference from mean
- **Range:** Max - Min

Sample Distribution

- **Histogram:** Visual distribution
- **Tally:** Simple counting
- **Skewness:** Asymmetry in data
- **Kurtosis:** "Tailedness" of the distribution

---

10. Binning

- Groups continuous variables into intervals.
- Types:
  1. **Equal-width:** Divides range into equal intervals.
  2. **Equal-frequency (Quantile):** Same number of samples per bin.
  3. **Custom:** User-defined.

---

11. Standardization

- Ensures all features contribute equally.
- **Z-Score:**
  - Formula: `z = (x - mean) / std_dev`

- **Min-Max:**
  - Formula: `(x - min) / (max - min)`

---

12. Outliers, Noise & Anomalies

Definitions

- **Outliers:** Values that deviate significantly from the rest.
  - **Global outlier:** Far from the entire dataset.
  - **Contextual outlier:** Unusual in a specific context.
- **Noise:** Random errors or variabilities in data.
- **Anomalies:** Unusual data points that may indicate error or rare events.

Detection & Handling

- Visualizations: Boxplots, scatterplots
- Statistical methods: Z-score, IQR methods, etc.
- Importance: Outliers can skew analyses and model results; sometimes they represent valuable rare events (e.g., fraud).