Data Problems

- Inconsistent column names
- Missing Data
- Outliers
- Duplicate Rows
- Untidy (Needing to process column)
- Wrong Data Type (signals unexpected data values)
- Typo and wrong values

Tidy Data

Three principles of tidy data:

- Column Represents Separate Variable
- Row Represents individual observation
- Observational unit forms table

Example of untidy data:

Name	Treatment A	Treatment B
Daniel	-	42
John	12	31
Jane	24	27

Tidy Version

Name	Treatment	Value
Jane	A	12
John	A	24
Daniel	B	42
John	B	31
Jane	B	27

- Tidy is better for analysis and untidy is better for reporting
- Data problem can be easily fixed in tidy data

Melt

- Tidy is better for analysis and untidy is better for reporting
- Data problem can be easily fixed in tidy data
- Puts data into common heading name
- Column size decreases & row size increases

Example:

```
melt = pd.melt(df, id_vars='name’, value_vars=['TreatA', 'TreatB’])

# Converts TreatA, TreatB into a single column
# If value_vars is not specified, all column except id_vars is melted
# Header of converted data can be provided by var_name & value_name
```

Exercise

- Generate DataFrame as below and convert to form having headers as **Country**, **Year**, **Gender**, **AgeGroup**, **Count**

Country	Year	M0-14	M15-24	F0-14	F15-24
Nepal	2000	25	75	10	11
China	2000	100	500	12	15
Afghanistan	2000	54	225	14	18

Pivot

- Opposite of melt
- Reshape from analysis friendly to reporting friendly
- Uses **pivot_table** for pivoting
- Default aggregation is average but we can specify other

Example:

```
df = pd.DataFrame({'Date': ['2025','2025','2026','2026'], 'Read':
['CO2', 'Rain', 'CO2', 'Rain'], 'Val': [0.1, 0.3, 0.12, 0.2]}

pvt = pd.pivot_table(df, index=['Date'], columns='Read',
values='Val')
# call .reset_index() to wipe index
```

Pivot

Other Arguments:

margins	➔	True (Adds total on row and column)
index	➔	Column to use as index
columns	➔	Whose data is to be pivoted by (can be list)
values	➔	Whose data is to be pivoted
fill_value	➔	Value to fill incase of null
aggfunc	➔	Aggregation to perform on duplicate

Exercise

- Convert data below into columns with fields: **Date**, **Day**, **Country**, **Case**, **Death**

Date	Day	case-nep	death-nep	case-ind	death-ind
2010	05	4	1	10	5
2010	06	3	2	20	8

Categorical

- We can load categorical column as categorical datatype
- We can even order categorical data making it as ordinal

Example:

```
df = pd.read_excel('air_bnb.xlsx')
print(df.room_type.nbytes)
print(df.room_type.value_counts())
df['room_type'] = df.room_type.str.title()
df['room_type'] = df.room_type.astype("category")
df['rt'] = pd.Categorical(df.room_type, [], ordered=True)
print(df.room_type.nbytes)
```

Data Value issue

- Extra characters in data value
  - $/Rs at beginning of monetary value
  - Thousands separator in monetary value
  - Leading/trailing space
  - Casing issue
  - Same value in multiple form. Female, Fe, F, Women, Girl
  - Typo in data entry. Kathmandu, Kathmundu
  - Data rule: dob in future, txn_date less than registered

Clean Value

- `df['price'] = df.price.str.strip("Rs$ ").replace(",", "")`
- `df['price'] = df.price.str.replace(r'\D+', '', regex=True)`
- `df['price'] = df.price.str.replace(r'[^0-9\.]', '', regex=True)`
- `df['gender'] = df.gender.replace({'Male': 'M', 'Boy': 'M'})`
- `df['gender'] = df.gender.map({'Male': 'M', 'Boy': 'M'})`

- .str.replace ➔ Replace individual character if present
- .replace ➔ Replace whole text. Doesn't look character wise
- .map ➔ Re-map current value. If key not in dict, stored as NaN

cut

- Converts continuous numeric variable into discrete group
- Segment based on Transaction, Grading
- Syntax: pd.cut(series, bins, labels, include_lowest)

Example:

```
import pandas as pd

df = pd.DataFrame({
   'name': ['Alice', 'Bob', 'Charlie', 'Diana', 'Edward', 'Fiona’],
   'age': [5, 15, 22, 35, 70, 12]})

bins = [0, 12, 17, 24, 64, 100]
labels = ['Child', 'Teen', 'Young Adult', 'Adult', 'Senior’]
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels,
    include_lowest=True)
```

Fuzzy Match

- No Direct match 0 - 1 but based on fuzzy logic
- For typo cleanup in data-set

Example:

```
df = pd.DataFrame({
   'city': [‘Kathmandu', ‘Bhoktapur', ‘kathmadu', ‘Lolitpur’],
   'age': [5, 15, 22, 35, 70, 12]})

standard_cities = ['Kathmandu', 'Bhaktapur', 'Lalitpur’]

df[‘cln’] = pd[‘city’].apply(clean_city)
```

Fuzzy Match

```
from fuzzywuzzy import process


def clean_city_name(city):
    match, score = process.extractOne(city, standard_cities)
    if score >= threshold:
        return match
    else:
        return city  # retain original if below threshold
```

EDA

- Organizing, Plotting and summarizing data
- pandas_profiling can be used for data profiling
- Common Operations:
  - .head(), .tail(), .columns, .index, .shape
  - .info
  - .isna, .isna().sum(), .isna().any()
  - .describe()
  - df.column.value_counts(dropna=False, sort=True, normalize=True)

Duplicate

```
duplicates = df[df.duplicated(subset=[‘c1’, ‘c2’])]
df.duplicated().sum()

data = data.drop_duplicates()
# Drops duplicate values from dataframe

data = data.drop_duplicates(subset=['col1', 'col2'])
# Drops duplicate data on the basis of column1 and column2
```

Visualizing Missing

```
na_value_count = df.isnull().sum()
missing_data = na_value_count[na_value_count > 0]
print(missing_data)
missing_data.plot(kind='bar', title='Missing', color="coral")
plt.xlabel('Column with missing data')
plt.ylabel('Missing Count')
plt.xticks(rotation=0)
plt.show()
```

Visualizing Missing

```
plt.figure(figsize=(12, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Data Heatmap")
plt.show()
```

Missing Value

```
df = pd.DataFrame({
        "id": [1, 2, 3, 4, 5],
        "course": ["python", None, "DS", "DS", "python"],
        "address": ["KTM", "PKR", None, "BRT", "KTM"],
        "age": [None, 34, 45, 23, 56],})

print(df)
data = data.dropna(how='any')   # all
df["course"] = df["course"].fillna("python")
df["course"] = df["course"].ffill()    # bfill for backward fill
df["age"] = df["age"].fillna(df.age.mode())    # mean, max, median
df.loc[df.age.isnull(), 'age'] = 25
```

Exercise