Data Engineering & Pre-processing

Session 2

Outline

- Data cleaning
- Computing Descriptive Statistics

Data Cleaning

Values in the data

- Before cleaning a dataset, it is a good idea to get a general feel for the data.
  - What are the expected values?

Numeric data

`df['col'].describe()`

```
data['Sales'].describe()
```

```
count      4646.000000
mean       2612.034869
std        9511.052145
min           0.000000
25%         392.000000
50%         672.000000
75%        1120.000000
max      345800.000000
```

Categorical data

`df['col'].describe()`

```
1  data['City'].describe()
```

```
count        4646
unique        222
top        BOSTON
freq          311
```

`df['col'].unique()`

```
1  data['City'].unique()
```

```
array(['BOYLSTON', 'SHREWSBURY',
       'WESTFORD', 'SALISBURY',
```

Datatypes (dtypes)

- What are the expected data types?

  `df.dtypes`

```
1  data.dtypes
```

```
Name          object
Address       object
City          object
State         object
State_name    object
Zip            int64
Sales          int64
Number_emp     int64
Empsiz        object
```

- Most common data types:
  - `float64`: floats
  - `int64`: integers
  - `object`: strings
  - `datetime64[ns]`: combined date/time info

- Modifying data types:
  - `df['col'].astype('key')`

    Converts the datatype of column *col*

Tweaking column names

- What are the column names?

  `df.columns`

```
1  data.columns
```

```
Index(['Name', 'Address', 'City', 'State', 'State_name', 'Zip', 'Sales',
       'Number_emp', 'Empsiz'],
      dtype='object')
```

  `df.rename()`

  Assigns new names to columns

```
data.rename(columns={'Number_emp':'Num_employes'}, inplace=True)
```

In-class activity 1

- Using the *dc_shipments* dataset, perform the following operations to explore and clean the data:
  - Identify the manufacturing plants where the shipments are coming from (unique values of column *plants*)
  - Identify the data types (dtypes) of each column in the dataset
  - Convert columns *dc* and *sku* data types to string (object). Verify this by calling dtypes again
  - Rename columns *ordered* and *delivered* by *qty_ordered* and *qty_delivered* respectively

Identifying missing data

- Detecting `na` (data not available) is pandas is simple:
  - `df.isna()`

    Returns a Boolean array (True if a value is missing) of the same dimensions as the DataFrame *df*.

    Could be applied to a whole DataFrame or single column.

```
data.isna()
```

	Name	Address	City	State	State_name	Zip	Sales	Number_emp	Empsiz
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False

```
data['Address'].isna()
```

```
0    False
1    False
```

  - `df.notna()`

    Opposite of `df.isna()`.

Identifying missing data (Cont.)

- How many `na` values are there in your dataset?
  - `df.isna().sum()`

    Returns the number of missing values by column in DataFrame *df*.

```
data.isna().sum()
```

```
Name          0
Address      35
City          0
State         0
State_name    0
Zip           0
Sales         0
Number_emp    0
Empsiz        0
```

Handling missing data

- Delete rows and columns with missing data.

`df.dropna()`

  - Rows: `df.dropna(inplace=True)`

```
data.dropna(inplace=True)
```

  - Columns: `df.dropna(axis=1,inplace=True)`

```
data.dropna(axis=1, inplace=True)
```

  - Drop only rows that contain missing data in specific columns: `df.dropna(subset=['colA','colB'],inplace=True)`

```
data.dropna(subset=['Address','City'], inplace=True)
```

Handling missing data (Cont.)

- Replace missing values.

```
df.fillna()
```

  - By a specific value:

```
df['col'].fillna(X,inplace=True)
```

    Replaces all `na` values of column *col* with X

  - By the column average:

```
df['col'].fillna(df['col'].mean(),inplace=True)
```

    Column average value

    Replaces all `na` values of column *col* with the mean

Replacing entries

- Wrong entries might be easy to spot in the dataset. Pandas helps in replacing them.
- Replacing values:

```
df.replace()
```

  - `df['col'].replace(a,b,inplace=True)`

    Replaces all `a` values in column *col* with `b`
  - `df['col'].replace([a,b],[c,d],inplace=True)`

    Replaces all `a` values with `c` and `b` values with `d` in column *col*

Handling duplicates

- Entries might be duplicated.
- Identifying duplicates:
  - `df.duplicated()`

    Checks for duplicates, returns Boolean array indicating if rows are duplicates (`True` is duplicate)
  - `df.drop_duplicates()`

    Drops all duplicates

In-class activity 2

- Using the *dc_shipments* dataset, perform the following operations to clean the data:
  - The column *plant* has a spelling error (west & West, Southeast & South east). Replace *west* by *West* and *Southeast* by *South east*. Verify this by calling .unique() again
  - How many duplicates are there in the dataset? Show those duplicated rows. Remove the duplicates and verify this by recounting the number of duplicates
  - How many *na* values by column are there in the dataset? Remove the rows with *na* values and confirm this by recounting the number of *na* values

Computing Descriptive Statistics

Statistical functions

- pandas is equipped with a set of common statistical functions.
- Syntax:

```
df['col'].describe()
```

- General statistical functions:
  - `df['col'].count()`
    Returns the number of non null-values in column *col*
  - `df['col'].sum()`
    Returns the sum of values of numeric column *col*

Statistical functions (Cont.)

- Measures of central tendency:
  - `df['col'].mean()`
    Returns the mean of numeric column *col*
  - `df['col'].median()`
    Returns the median of numeric column *col*
  - `df['col'].mode()`
    Returns the mode of column *col*

Statistical functions (Cont.)

- Measures of dispersion:
  - `df['col'].min()/df['col'].max()`
    Returns the lowest/highest value in numeric column *col*
  - `df['col'].idxmin()/df['col'].idxmax()`
    Returns the row (index) at which min/max is obtained
  - `df['col'].var()`
    Returns the sample variance of values of numeric column *col*
  - `df['col'].std()`
    Returns the sample standard deviation of values of column *col*

In-class activity 3

- Use the CO2_emissions_2017 dataset to identify the most polluting vehicle.
- To answer this question, write a Python program that perform the following operations:
  - Create a column with the average C02 emissions
  - Create a column with the standard deviation of CO2 emissions
  - Create a column calculating the standardized C02 emissions: (value-average)/standard deviation
  - Identify the vehicle (ID) with the largest standardized C02 emissions. What is the maximum value?

References

- McKinney, W., Pydata Development Team, 2019. pandas: powerful Python data analysis toolkit. Python package.

Thank you for attention!

Ilya Jackson, Ph.D.
✉ ilyajack@mit.edu