DATA
SOCI
ETY

Data wrangling in Python - Data wrangling with Pandas - 5

One should look for what is and not what he thinks should be. (Albert Einstein)

Chat question

- Based on what you know about each library, what do you think are the **major differences** between **NumPy** and **Pandas**?
- **Share your thoughts** using the chat or aloud
- If you want some help getting started, take a few minutes to skim through ***this article***

Module completion checklist

Objective	Complete
Create subsets and clean data using Pandas	
Understand how to reshape data using Pandas	

Data wrangling and exploration

- Remember, a data scientist must be able to:
  - **Wrangle** the data (gather, clean, and sample data to get a suitable dataset)
  - **Manage** the data for easy access by the organization
  - **Explore** the data to generate a hypothesis
- By this point, we have started to practice each of these skills

- We will work with the dataset to practice in greater depth and detail
- We will be:
  - Cleaning the dataset to address missing values
  - Wrangling the data for the purpose of visualizing the data and identifying patterns

Subsetting data

- We will explore a subset of this dataset, which includes the following variables:
  - bmi
  - avg_glucose_level
  - age
  - heart_disease
  - hypertension
  - Residence type
  - ever married
- We are choosing these variables because they illustrate the concepts best
- However, you should be able to work with (and visualize) **all** of your data

Creating a subset

- Let’s create a subset of the data called `df_subset`
- This subset will contain **only** the variables we need

```
df_subset = df[['bmi', 'avg_glucose_level', 'age', 'heart_disease', 'hypertension', 'stroke',
'Residence_type', 'ever_married', 'smoking_status', 'gender', 'work_type']]
print(df_subset.head())
```

```
    bmi  avg_glucose_level   age  ...   smoking_status  gender      work_type
0  36.6             228.69  67.0  ...  formerly smoked    Male        Private
1   NaN             202.21  61.0  ...     never smoked  Female  Self-employed
2  32.5             105.92  80.0  ...     never smoked    Male        Private
3  34.4             171.23  49.0  ...           smokes  Female        Private
4  24.0             174.12  79.0  ...     never smoked  Female  Self-employed

[5 rows x 11 columns]
```

Data prep: clean NAs

- Depending on the **subject matter**, missing values in our dataset might signify different things
- We can handle NAs in our data in a number of ways:
  - drop columns that contain any `NA`s
  - drop columns with a certain % of `NA`s
  - impute missing values
  - convert column with missing values to categorical data type

Data prep: clean NAs (cont’d)

- Let’s look at the count of NAs by column first:

```
print(df_subset.isnull().sum())
```

```
bmi                    201
avg_glucose_level        0
age                      0
heart_disease            0
hypertension             0
stroke                   0
Residence_type           0
ever_married             0
smoking_status        1544
gender                   0
work_type                0
dtype: int64
```

Data cleaning: NAs

- If a variable has more than 50% NAs, we could just drop this column
- However, if it is less than 50%, we'll keep it, and **impute** missing values using the **mean** of the column
- There isn't a mathematical method for the precise percentage of NAs that we find acceptable
- That's why your **subject matter expertise** is so important!

Data cleaning: NAs (cont'd)

- We will now impute the numerical columns containing NAs with their mean values

```
# Set the DataFrame equal to the imputed dataset.
df_subset[['bmi', 'avg_glucose_level', 'age']] = df_subset[['bmi', 'avg_glucose_level',
'age']].fillna(df_subset.mean())
# Check how many values are null in the numerical variables.
print(df_subset.isnull().sum())
```

```
bmi                     0
avg_glucose_level       0
age                     0
heart_disease           0
hypertension            0
stroke                  0
Residence_type          0
ever_married            0
smoking_status       1544
gender                  0
work_type               0
dtype: int64
```

Module completion checklist

Objective	Complete
Create subsets and clean data using Pandas	✔
Understand how to reshape data using Pandas	

Data reshaping: wide vs. long

- When we talk about data *reshaping*, what we usually mean is converting between what is called either **wide** or **long** data format
    - **Wide** data is much more visually digestible, which is why you're likely to come across it if you are using data from some type of report
    - **Long** data is much easier to work with in Pandas, and generally speaking in most data analysis and plotting tools
- **Wide** data often appears when the values are some type of aggregate (we will use `mean` of groups)
- Let's make a `DataFrame` with two rows and six columns, representing a typical wide DataFrame

Prepare data: group and summarize

- Now that we know how to group and summarize data, let's create a summary dataset that would include the following:
  - Group data by a categorical variable with a low number of levels
  - Mean value computed on the grouped data that includes the following variables:
    - bmi
    - avg_glucose_level
    - age

Prepare data: group and summarize (cont’d)

- For the purpose of demonstration, we use the original DataFrame `df` to identify the grouping column
- We then use this column to perform the `groupby()` operation and find the mean of the columns present in `df_subset`

```
col_dict = df_subset.nunique().to_dict()
grouping_col = min(col_dict, key=col_dict.get)
# Group data by variable with min levels.
grouped = df_subset.groupby(grouping_col)
```

```
# Compute mean on the listed variables using the grouped data.
df_grouped_mean = grouped.mean()[['bmi', 'avg_glucose_level', 'age']]
print(df_grouped_mean)
```

```
                     bmi  avg_glucose_level        age
heart_disease
0              28.821693         104.396494  41.801407
1              30.146293         136.818768  68.188406
```

Prepare data: group and summarize (cont’d)

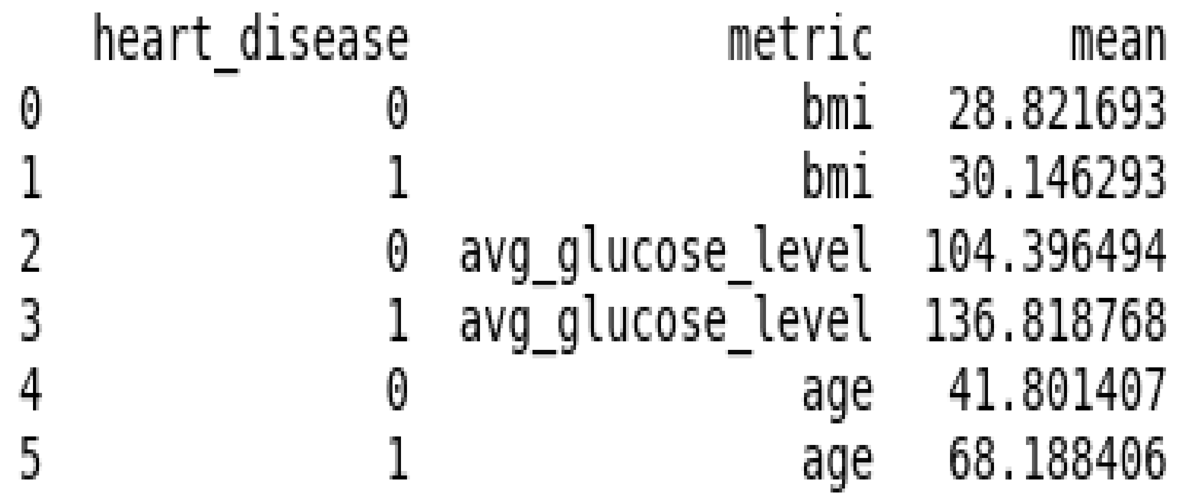
```
# Reset index of the dataset.
df_grouped_mean = df_grouped_mean.reset_index()
print(df_grouped_mean)
```

```
   heart_disease        bmi  avg_glucose_level        age
0              0  28.821693         104.396494  41.801407
1              1  30.146293         136.818768  68.188406
```

- The reason we call this DataFrame **wide** is because each variable has its own column
- It makes the table easier to present, but is inconvenient to run analyses on or visualize

Why long?

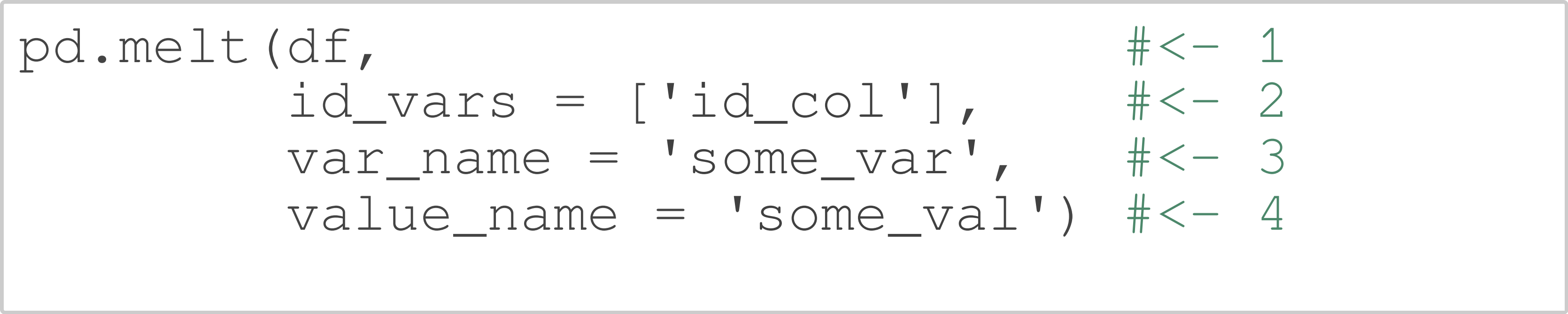
- Now let’s convert this wide data to the **long** format
  - We are going to leave the `categorical` variable and the mean values as their own columns
  - All of our other variables will appear as a single metric column
- This format is convenient to work with when we run an analysis and plot the data



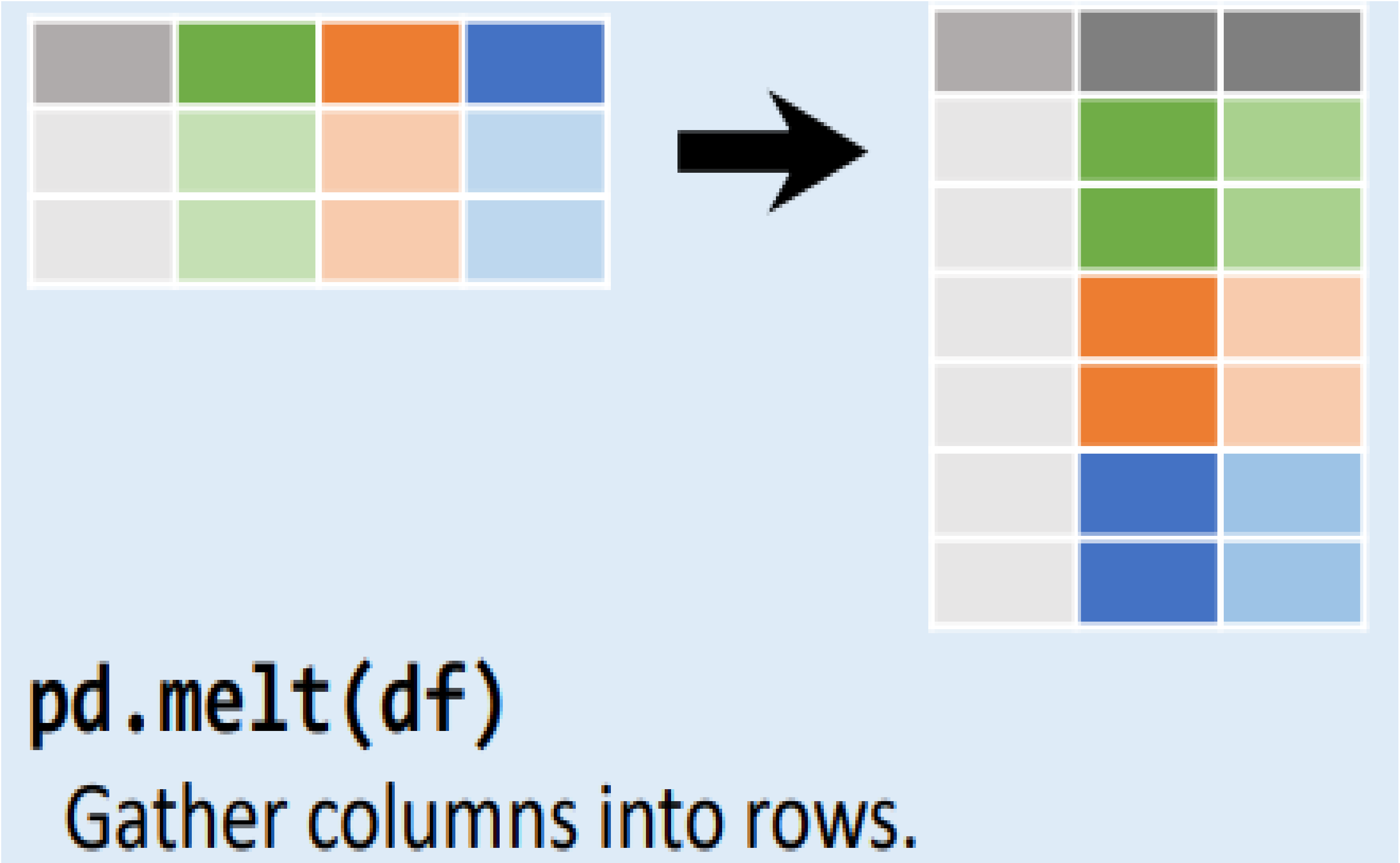
```
   heart_disease           metric        mean
0              0              bmi   28.821693
1              1              bmi   30.146293
2              0  avg_glucose_level  104.396494
3              1  avg_glucose_level  136.818768
4              0              age   41.801407
5              1              age   68.188406
```

Wide to long format: melt

- To **convert** from wide to long format, we use the Pandas `melt` function with the following arguments:
  i. Wide DataFrame
  ii. Variable(s) that will be preserved as the `ids` of the data (i.e. like categorical variables)
  iii. Name of the variable that will now contain the column names from the wide data we want to melt together
  iv. Name of the column that will contain respective values corresponding to the melted columns



```
pd.melt(df,                                #<- 1
        id_vars = ['id_col'],              #<- 2
        var_name = 'some_var',             #<- 3
        value_name = 'some_val')           #<- 4
```



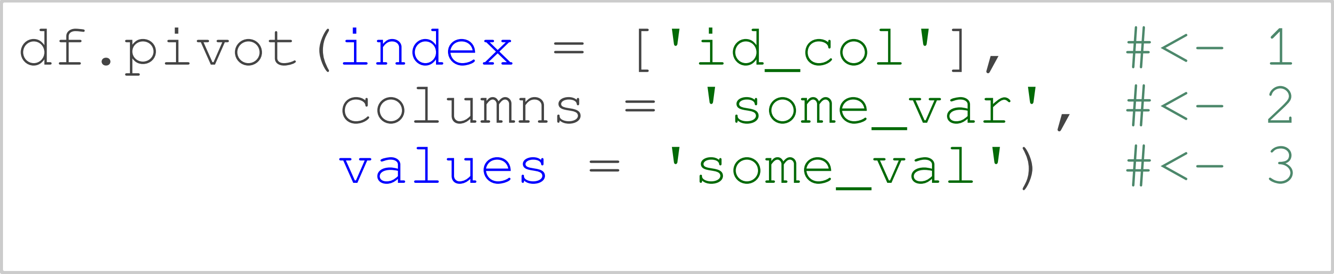
pd.melt(df)
Gather columns into rows.

Wide to long format: melt (cont'd)

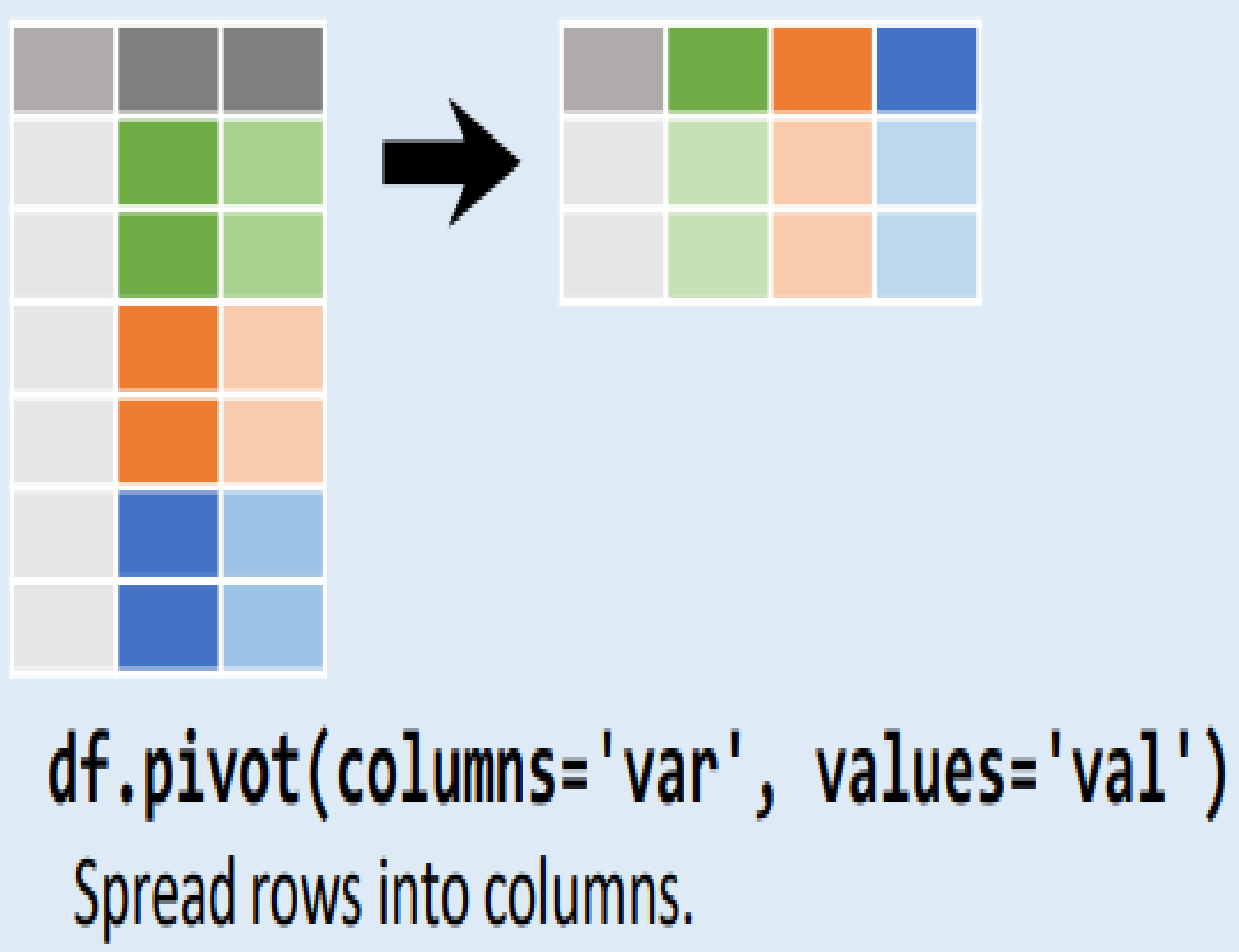
```
# Melt the wide data into long.
df_grouped_mean_long = pd.melt(df_grouped_mean,       #<- wide dataset
                                  id_vars = [grouping_col],     #<- identifying variable
                                  var_name = 'metric',      #<- contains col names of wide data
                                  value_name = 'mean')      #<- contains values from above columns
print(df_grouped_mean_long)
```

```
   heart_disease             metric        mean
0              0                bmi   28.821693
1              1                bmi   30.146293
2              0  avg_glucose_level  104.396494
3              1  avg_glucose_level  136.818768
4              0                age   41.801407
5              1                age   68.188406
```

Long to wide format: pivot



```
df.pivot(index = ['id_col'],    #<- 1
         columns = 'some_var',  #<- 2
         values = 'some_val')   #<- 3
```



df.pivot(columns='var', values='val')

Spread rows into columns.

- We can **convert** the long data **back to wide** format with the `.pivot()` method
  - The `index` argument refers to what values *will become* the `ids` in the new DataFrame
  - The `columns` argument refers to the column in which its values will be converted to column names
  - Lastly, we supply the `values` argument to fill in the values of the wide data

Long to wide format: pivot (cont’d)

```
# Melt the long data into wide.
df_grouped_mean_wide = df_grouped_mean_long.pivot(
                                                    index = [grouping_col],    #<- identifying 
variable
                                                    columns = 'metric', #<- col names of wide data
                                                    values = 'mean')    #<- values from above 
columns
print(df_grouped_mean_wide)
```

```
metric                age  avg_glucose_level        bmi
heart_disease
0               41.801407         104.396494  28.821693
1               68.188406         136.818768  30.146293
```

Knowledge check

Link: ***https://forms.gle/apk1hxHzvuut3iA97***

Exercise

You are now ready to try Tasks 17-20 in the Exercise for this topic

Module completion checklist

Objective	Complete
Create subsets and clean data using Pandas	✔
Understand how to reshape data using Pandas	✔

Data Wrangling with Pandas: Topic summary

In this part of the course, we have covered:

- Pandas use cases and basic operations
- DataFrame definition and manipulation

Congratulations on completing this module!