SCALER

PANDAS CHEAT-SHEET

1 INSTALLING AND IMPORTING

Installing	`pip install pandas`
Importing Convention	`import pandas as pd`

2 READING AND WRITING DATA

Reading data	`df = pd.read_csv (path = 'filename.csv')`
	
	`# can extend for json, excel types too using pd.read_json/pd.read_excel, etc.`

Writing data	`df.to_csv('filename.csv')`
	
	`# can extend for json, excel and such too using df.to_json/df.to_excel, etc.`

3 SERIES AND DATAFRAMES

• Creating a series

```
pd.Series(['a', 'b', 'c'])
```

• Creating a dataframe

Row Oriented	`pd.DataFrame([['a', 1], ['b', 2]], columns=['name', 'id'])`
Column Oriented	`pd.DataFrame({'name': ['a', 'b'], 'id':[1, 2]})`

	name	id
0	a	1
1	b	2

4 INFO EXTRACTION

Shape (Return a tuple representing the dimensionality of the DataFrame.)	`df.shape` `#e.g.-(2,3) for 2 rows and 3 columns`
Head (first n rows, default 5)	`df.head(n)`
Tail (last n rows, default 5)	`df.tail(n)`
Info (return info of all columns)	`df.info()`
Describe (gives statistical information of data)	`df.describe()`

5 ACCESSING

• Direct accessing columns and rows, as well as both together

Accessing a row	`df.loc[ei]`
	`#(ei here is explicit index)`
	`df.iloc[ii]`
	`#(ii here is implicit index)`

Accessing a column	`df['column_name']`
	`#for single column`
	`df[['col1', 'col2']]`
	`#for multiple columns`

• Slicing

Row	`df.loc[1:3]`
	`(1 and 3 are the explicit indices here)`
	`Or`
	`df.iloc[2:4] (2 and 4 are the implicit indices here)`

Column	`df.loc[:, 'a':'b']`
Both row and columns	`df.loc[1:3, 'a':'b'] (1 and 3 are explicit indices here)`

• Feature exploration (masking, filtering)

Masking Creates a mask based on our required condition	`df['col''] > value` `E.g. df['age'] > 30`

Filtering
Filters data based on conditions
	`df.loc[(df['col1'] == val1) & (data['col2'] ==val2)]`
	
	`E.g.`
	`df.loc[(df['month'] == 'January') & (df['year']=='2022')]`
	`# filters out data for january 2022`

6 DATAFRAME MANIPULATION

• Adding a new row/column

Row	`df.loc[explicit_row_num] = ['a', 1]`
	`E.g.`
	`df.loc[len(df.index)] = ['a', 1]`
	`# this will add a row at the end of the dataframe`

Column	`df['new_col']=data`

• Deleting a new row/column

Row	`df.drop(labels=None, axis=0)`
	
	`E.g.`
	`df.drop(3, axis=0)`
	`# Here 3 is the explicit index, axis=0 is for row`

Column	`df.drop('col_name', axis=1)`

• Renaming a column

Row	`df.index = new_indices`
Column	`df.rename({'old_name':'new_name', axis=1})`

• Duplicates and dropping duplicates

- find duplicate rows

`df.duplicated`(subset=None, keep='first')
 # subset can be used to specify certain column(s) for identifying the duplicates
 # keep determines which duplicates to mark

1. **first** : marks all duplicates as true except for the first occurrence.
2. **last** : marks all duplicates as true except for the last occurrence.
3. **False** : marks all duplicates as true

returns a boolean series for each duplicate row marked as true

- drop duplicate values

`df.drop_duplicates`(subset=none, keep='first')

 # parameters have the same meaning as in df.duplicated, except here it will drop the rows marked duplicate

7 OPERATIONS

• Sorting

```
df.sort_values(['col1'], ascending=[True])
```

• Built in ops

built in ops such as mean, min, max, etc.
e.g., `df['col1'].min(), df['col1'].count(),` etc.

• Apply

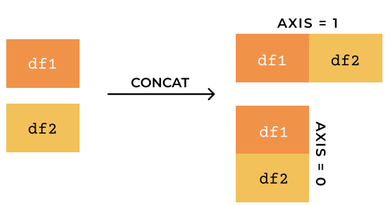
applies a function along one of the axis of the dataframe
df['col'].apply(function)

e.g.

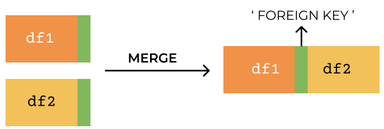
```
data[['revenue', 'budget']].apply(np.sum, axis=1)
#sums values of revenue and budget across each row
```

8 JOINS

• concat
`pd.concat`([df1, df2], axis = 0) (for concatenating horizontally, change axis = 1)



• merge
`df1.merge`(df2, on='foreign_key', how='type_of_join')
- optional -> left_on and right_on
- eg. df1.merge(df2, on='id', how='inner')



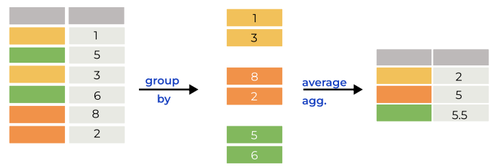
9 GROUPBY

Grouping based on a single aggregate	`df.groupby('group_col_name')['col(s)'].aggregate_function()`
	
	`E.g.`
	`df.groupby('director_name')['title'].count()`
	`# Finds number of titles per director`

Grouping based on multiple aggregates	`df.groupby(['group_col_name'])['col'].aggregate(['func1', 'func2'])`
	
	`E.g.`
	`df.groupby(['director_name'])["year"].aggregate(['min', 'max'])`
	`# Finds first and recent year of movies made by all directors`

Group based filtering	`df.groupby('group_col_name').filter(boolean array based on condition)`
	
	`E.g.`
	`data.groupby('director_name').filter(lambda x: x["budget"].max() >= 100)`
	
	`# This filters all rows of those directors whose maximum budget is greater than 100 million)`

Group based apply	`df.groupby('group_col_name').apply(function)`
	
	`E.g.`
	`def func(x):`
	`x["risky"] = x["budget"] - x["revenue"].mean() >= 0`
	`return x`
	`data_risky = data.groupby ("director_name").apply(func)`
	
	`# Finds movies whose budget is higher than its director's average revenue`



10 CLEANING OUR DATA

• None and nan

- "NaN" is for columns with numbers as their values
- "None" is for columns with non-number entries(e.g. String, object type, etc.)
- Can check for null values using "isna()"
  - E.g.
  - **df.isna()** # returns the dataframe with True/False for null values in the respective element's position
  - **df.isna().sum()** # returns number of null values per column. Can modify with df.isna().sum(axis=1) for each row's null count
  - **df.isna().sum().sum()** # returns total number of null values

• Filling null values

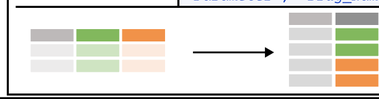
```
df.fillna(n) # fills null values with value 'n'
```

• Dropping null values

`df.dropna`(axis = 0)
Default axis=0, use 1 for columns
Drops rows/columns with even a single missing value

11 DATA RESTRUCTURING

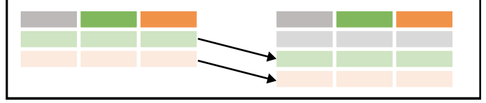
Melt
Convert dataframe from wide to long format
	`pd.melt(df, id_vars=['list of columns'])`
	`E.g.`
	`pd.melt(data, id_vars=['Date', 'Parameter', 'Drug_Name'])`



Pivot
Opposite of melt, converts dataframe from long to wide format
 	
Outputs a multi-index dataframe
	`df.pivot(index=['list of columns], columns='col_name', values='col_name')`
	
	`E.g.`
	`data_melt.pivot(index=['Date','Drug_Name','Parameter'], columns = 'time', values='reading')`

Cut
Bins continuous data into categorical groups
	`df['new_cat_column']=pd.cut (df['continous_col'],bins=bin_values, labels=label_values)`
	
	`E.g.`
	`data_tidy['temp_cat'] = pd.cut(data_tidy['Temperature'], bins=temp_points, labels=temp_labels)`

Shift
Shifts the values of rows/columns
	`df['col'].shift(periods=n, axis=0)`
	
	`E.g.`
	`df["Marks"].shift(periods = 1, axis = 0)`



12 MISC TOPICS

• Datetime

- convert to datetime object: pd.to_datetime(df['col'])

• extracting information

df['col'][0].year	Extracts the year for the 0th index value Here 0 is the implicit index Use .month and .day for the respective data
df['col'].dt.year	Extracts the year for whole columns (all the datetime values)
df['col'][0].strftime('%M%Y')	Formats the select data (0th index datetime value here) into the required data time format (month and year in this case)

• String functions

we can use .str to apply string functions to any column
df['col'].str.function()

e.g.
i. data_tidy['date'].str.split('-')
this will split the "date" column into elements separated by "-"
ii. data_tidy.loc[data_tidy['drug_name'].str.contains ('hydrochloride')]
will filter out rows containing the string "hydrochloride"