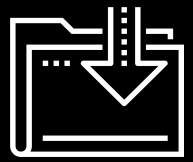
Exploring Pandas

Data Boot Camp

Lesson 4.2



Class Objectives

By the end of today's class, you will be able to:

Navigate through DataFrames using `loc` and `iloc`.

Filter and slice Pandas DataFrames.

Create and access Pandas `groupby` objects.

Sort DataFrames.

Instructor Demonstration

Exploring Data with `loc` and `iloc`

Programmers can easily collect specific rows and columns of data from a DataFrame by using the `loc` and `iloc` methods.

Exploring Data with `loc` and `iloc`

- `loc` returns data based on an index of labels/strings
- `loc` is limited to string types and cannot be used on a numerical index. As an alternative solution, you can use the `df.set_index()` function, passing in the desired column header for the index.
- Instead of using labels, `iloc` uses integer-based indexing for selection by position.

```
# Set new index to Species
df = original_df.set_index("Species")
df.head()
```

Species	Scientific Name	Common Name	Threatened status	Kingdom	Class	Family	Genus	Species Author	Profile	extra
cinerea	Neophoca cinerea	Australian Sea-lion, Australian Sea Lion	Vulnerable	Animalia	Mammalia	Otariidae	Neophoca	(Peron,1816)	http://www.environment.gov.au/cgi-bin/sprat/pu...	2 Oc
leonina	Mirounga leonina	Southern Elephant Seal	Vulnerable	Animalia	Mammalia	Phocidae	Mirounga	(Linnaeus,1758)	http://www.environment.gov.au/cgi-bin/sprat/pu...	2 Oc
novaehollandiae	Pseudomys novaehollandiae	New Holland Mouse, Pookila	Vulnerable	Animalia	Mammalia	Muridae	Pseudomys	(Waterhouse, 1843)	http://www.environment.gov.au/cgi-bin/sprat/pu...	2 Oc
borealis	Balaenoptera borealis	Sei Whale	Vulnerable	Animalia	Mammalia	Balaenopteridae	Balaenoptera	Lesson, 1828	http://www.environment.gov.au/cgi-bin/sprat/pu...	2 Oc

Exploring Data with `loc` and `iloc`

- Both `loc` and `iloc` use brackets that contain the desired rows, followed by a comma and the desired columns.
- For example, `loc[“leonina”, “Family”]` or `iloc[1,5]`

```
# Get the data contained within the "leonina" row and the "Family" column
leonina_family = df.loc["leonina", "Family"]
print(f"Using loc: {leonina_family}")

also_leonina_family = df.iloc[1, 5]
print(f"Using iloc:{also_leonina_family}")
```

```
Using loc: Phocidae
Using iloc:Phocidae
```

Exploring Data with `loc` and `iloc`

Both methods allow us to select a range of columns and rows by providing a list.

We can also use a colon to tell Pandas to look for a range.

```
# Get the first five rows of data and the columns from "Common Name" to "Threatened status"
# The problem with using "Species" as the index is that the values are not unique so duplicates are returned
# If there are duplicates and loc is being used, Pandas will return an error
cinerea_to_musculus = df.loc[["cinerea", "leonina", "novaehollandiae", "borealis",
                              "musculus"], ["Common Name", "Threatened status"]]
print(f"Using loc:{cinerea_to_musculus}")

print()

# Using iloc will not find duplicates since a numeric index is always unique
also_cinerea_to_musculus = df.iloc[0:5, 1:3]
print(f"Using iloc:{also_cinerea_to_musculus}")
```

```
Using loc:                                                  Common Name  \
Species
cinerea                   Australian Sea-lion, Australian Sea Lion
leonina                                     Southern Elephant Seal
novaehollandiae                          New Holland Mouse, Pookila
novaehollandiae  Cape Barren Goose (south-western), Recherche C...
novaehollandiae                                Masked Owl (northern)
novaehollandiae           Tiwi Masked Owl, Tiwi Islands Masked Owl
novaehollandiae                     Tasmanian Emu, Emu (Tasmanian)
novaehollandiae                             Masked Owl (Tasmanian)
borealis                                                 Sei Whale
musculus                                                Blue Whale

                Threatened status
Species
cinerea               Vulnerable
leonina               Vulnerable
novaehollandiae       Vulnerable
novaehollandiae       Vulnerable
novaehollandiae       Vulnerable
novaehollandiae       Endangered
novaehollandiae          Extinct
```

Exploring Data with `loc` and `iloc`

By passing in a colon by itself, `loc` and `iloc` will select all rows or columns depending on where the colon is placed in relation to the comma.

```
# The following will select all rows for columns `Common Name` and `Threatened Status`
df.loc[:, ["Common Name", "Threatened status"]].head()
```

Species	Common Name	Threatened status
cinerea	Australian Sea-lion, Australian Sea Lion	Vulnerable
leonina	Southern Elephant Seal	Vulnerable
novaehollandiae	New Holland Mouse, Pookila	Vulnerable
borealis	Sei Whale	Vulnerable
musculus	Blue Whale	Endangered

`loc` and `iloc` can be used to conditionally filter rows of data based on the values within a column.

Exploring Data with `loc` and `iloc`

- Instead of passing a list of indexes, we can use a logic statement.
- If multiple conditions should be checked, `&` and `|` may also be added into the logic test as representations of `and` and `or`.

```
# Multiple conditions can be set to narrow down or widen the filter
only_endangered_and_critical = df.loc[(df["Threatened status"] == "Endangered") | (
    df["Threatened status"] == "Critically Endangered"), :]
only_endangered_and_critical
```

Species	Scientific Name	Common Name	Threatened status	Kingdom	Class	Family	Genus	Species Author	Profile	e
musculus	Balaenoptera musculus	Blue Whale	Endangered	Animalia	Mammalia	Balaenopteridae	Balaenoptera	(Linnaeus, 1758)	http://www.environment.gov.au/cgi-bin/sprat/pu...	
australis	Eubalaena australis	Southern Right Whale	Endangered	Animalia	Mammalia	Balaenidae	Eubalaena	(Desmoulins, 1822)	http://www.environment.gov.au/cgi-bin/sprat/pu...	
pedunculatus	Zyzomys pedunculatus	Central Rock-rat, Antina	Critically Endangered	Animalia	Mammalia	Muridae	Zyzomys	(Waite,1896)	http://www.environment.gov.au/cgi-bin/sprat/pu...	
shortridgei	Pseudomys shortridgei	Heath Mouse, Dayang, Heath Rat	Endangered	Animalia	Mammalia	Muridae	Pseudomys	(Thomas, 1907)	http://www.environment.gov.au/cgi-bin/sprat/pu...	
fumeus	Pseudomys fumeus	Smoky Mouse, Konoom	Endangered	Animalia	Mammalia	Muridae	Pseudomys	Brazenor,1934	http://www.environment.gov.au/cgi-bin/sprat/pu...	

Activity: Good Movies

In this activity, you will create an application that searches through IMDb data to find only the top-rated movies.

Suggested Time:

20 minutes

Time’s Up! Let’s Review.

Instructor Demonstration

Cleaning Data

When dealing with massive datasets, it’s almost inevitable that we’ll encounter duplicate rows, inconsistent spelling, and missing values.

Cleaning Data

`del <DataFrame>[<columns>]`

```
# Preview of the DataFrame
# Note that OCCUP_INDEX appears to be duplicating OCCUP
employment_data_df.head()
```

	NAME	AGE	OCCUP	YEAR	NAME_RANK	EMPLOY_DATE	EMPLOYER	OCCUP_INDEX	RECORD_URL
0	Crowther, Edward Lodewyk	33.0	Doctor of Medicine	1878	Commanding Officer	23 Apr 1878	Southern Volunteer Artillery	Doctor of Medicine	https://stors.tas.gov.au/NI/1517087
1	Crowther, William L	63.0	Doctor of Medicine	1878	Surgeon Major	10 May 1878	Southern Volunteer Artillery	Doctor of Medicine	https://stors.tas.gov.au/NI/1517088
2	Roblin, Thomas	50.0	Curator of Museum	1878	Lieutenant	23 Apr 1878	Southern Volunteer Artillery	Curator of Museum	https://stors.tas.gov.au/NI/1517089
3	Lewis, D	NaN	Merchant	1878	Major Paymaster	23 Apr 1878	Southern Volunteer Artillery	Merchant	https://stors.tas.gov.au/NI/1517090
4	Green, William Patrick	NaN	Gentleman	1878	Quartermaster Captain	16 Aug 1878	Southern Volunteer Artillery	Gentleman	https://stors.tas.gov.au/NI/1517091

```
# Delete column we don't want
del employment_data_df['OCCUP_INDEX']
employment_data_df.head()
```

	NAME	AGE	OCCUP	YEAR	NAME_RANK	EMPLOY_DATE	EMPLOYER	RECORD_URL
0	Crowther, Edward Lodewyk	33.0	Doctor of Medicine	1878	Commanding Officer	23 Apr 1878	Southern Volunteer Artillery	https://stors.tas.gov.au/NI/1517087
1	Crowther, William L	63.0	Doctor of Medicine	1878	Surgeon Major	10 May 1878	Southern Volunteer Artillery	https://stors.tas.gov.au/NI/1517088
2	Roblin, Thomas	50.0	Curator of Museum	1878	Lieutenant	23 Apr 1878	Southern Volunteer Artillery	https://stors.tas.gov.au/NI/1517089
3	Lewis, D	NaN	Merchant	1878	Major Paymaster	23 Apr 1878	Southern Volunteer Artillery	https://stors.tas.gov.au/NI/1517090
4	Green, William Patrick	NaN	Gentleman	1878	Quartermaster Captain	16 Aug 1878	Southern Volunteer Artillery	https://stors.tas.gov.au/NI/1517091

Cleaning Data

`count()`
`<DataFrame>.dropna(how='any')`

```
# Identify incomplete rows
employment_data_df.count()
```

```
NAME           4325
AGE             811
OCCUP          3739
YEAR           4172
NAME_RANK       675
EMPLOY_DATE     852
EMPLOYER        877
RECORD_URL     4325
dtype: int64
```

```
# Drop all rows with missing information
employment_data_df = employment_data_df.dropna(how='any')
```

```
# Verify dropped rows
employment_data_df.count()
```

```
NAME           613
AGE            613
OCCUP          613
YEAR           613
NAME_RANK      613
EMPLOY_DATE    613
EMPLOYER       613
RECORD_URL     613
dtype: int64
```

Cleaning Data

`value_counts()`
`replace()`

```
# Display an overview of the OCCUP column
employment_data_df['OCCUP'].value_counts()
```

```
Clerk                          85
Labourer                       41
Bootmaker                      40
Carpenter                      30
Blacksmith                     23
                               ..
Barrister at Law                1
Coach Trimmer                   1
Fitter and turner on railway    1
Brewer                          1
Engine Driver                   1
Name: OCCUP, Length: 170, dtype: int64
```

```
# Clean up OCCUP category. Replace 'Laborer' with 'Labourer',
# 'Stone Mason' with 'Stonemason', 'Boot Maker' with 'Bootmaker'
# 'Coachtrimmer' with 'Coach Trimmer', and 'None' with 'None at present'
employment_data_df['OCCUP'] = employment_data_df['OCCUP'].replace({'Laborer': 'Labourer',
                                    'Stone Mason': 'Stonemason',
                                    'Boot Maker': 'Bootmaker',
                                    'Coachtrimmer': 'Coach Trimmer',
                                    'None': 'None at present'})
```

```
# Verify clean-up.
employment_data_df['OCCUP'].value_counts()
```

```
Clerk                          85
Labourer                       50
Bootmaker                      42
Carpenter                      30
Blacksmith                     23
                               ..
```

Activity: Hong Kong LPG Appliances

In this activity, you will take an LPG appliance dataset from Hong Kong, and clean it so that the DataFrame is consistent, and that there are no rows with missing data.

Suggested Time:

20 minutes

Time’s Up! Let’s Review.

Activity: Pandas Recap and Data Types

In this activity, we will recap what has been covered in Pandas up to this point.

Suggested Time:

15 minutes


Break

Instructor Demonstration

Pandas Grouping

`.groupby()` is a simpler method for filtering data.

Pandas Grouping

To split the DataFrame into multiple groups and group by local government area, we use `df.groupby([<Columns>])`.

The `.groupby()` method returns a `groupby` object that can only be accessed by using a data function on it.

```
# Count how many inaccessible road stops are in each local government area
grouped_lg_df = road_stops_df.groupby(["LOCAL_GOVERNMENT_NAME"])

print(grouped_lg_df)

grouped_lg_df.count().head(10)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7feca09d4f10>
```

LOCAL_GOVERNMENT_NAME	COMMON_USAGE_NAME	RESPONSIBILITY_AREA	REST_AREA_TYPE	STAY_24_HOUR	OWNERSHIP	SURFACE	SURFACE_TYPE
Albany (C)	25	25	25	25	25	25	25
Armadale (C)	4	4	4	4	4	4	4
Ashburton	70	70	70	70	70	70	70
Augusta - Margaret River	11	11	11	11	11	11	11
Beverley	8	8	8	8	8	8	8
Boddington	9	9	9	9	9	9	9
Boyup Brook	6	6	6	6	6	6	6
Bridgetown - Greenbushes	3	3	3	3	3	3	3
Brookton	6	6	6	6	6	6	6
Broome	8	8	8	8	8	8	8

Pandas Grouping

The `pd.DataFrame()` method makes it possible to create new DataFrames by using only `groupby` data.

A DataFrame can also be created by selecting a single Series from a `groupby` object and passing it in as the values for a specified column.

```
# Save toilets and tables sums as series
lg_toilets = grouped_lg_df["NUMBER_OF_TOILETS"].sum()
lg_tables = grouped_lg_df["NUMBER_OF_TABLES"].sum()
lg_tables.head()
```

```
LOCAL_GOVERNMENT_NAME
Albany (C)                  9
Armadale (C)                0
Ashburton                   9
Augusta - Margaret River    0
Beverley                    2
Name: NUMBER_OF_TABLES, dtype: int64
```

```
# Create a new DataFrame using count and total toilets, and tables
lg_summary_df = pd.DataFrame({"Number of Road Stops with Accessibility Issues": lg_counts,
                              "Total Toilets": lg_toilets,
                              "Total Tables": lg_tables})
lg_summary_df.head()
```

	Number of Road Stops with Accessibility Issues	Total Toilets	Total Tables
Albany (C)	25	0	9
Armadale (C)	4	0	0
Ashburton	70	7	9
Augusta - Margaret River	11	0	0
Beverley	8	0	2

Pandas Grouping

It's also possible to perform a `df.groupby()` method on multiple columns by passing two or more column references into the list parameter.

```
# It is also possible to group a DataFrame by multiple columns
# This returns an object with multiple indexes, however, which can be harder to deal with
grouped_lg_surface = road_stops_df.groupby(["LOCAL_GOVERNMENT_NAME","SURFACE"])

grouped_lg_surface.count().head(10)
```

		COMMON_USAGE_NAME	RESPONSIBILITY_AREA	REST_AREA_TYPE	STAY_24_HOUR	OWNERSHIP	SURFACE_TYPE
LOCAL_GOVERNMENT_NAME	SURFACE						
Albany (C)	Surfaced	9	9	9	9	9	9
	Unsurfaced	16	16	16	16	16	16
Armadale (C)	Unsurfaced	4	4	4	4	4	4
Ashburton	Surfaced	26	26	26	26	26	26
	Unsurfaced	44	44	44	44	44	44
Augusta - Margaret River	Surfaced	8	8	8	8	8	8
	Unsurfaced	3	3	3	3	3	3
Beverley	Surfaced	1	1	1	1	1	1
	Unsurfaced	7	7	7	7	7	7
Boddington	Surfaced	4	4	4	4	4	4

Pandas Grouping

A new DataFrame can be created from a `groupby` object.

```
# Converting a GroupBy object into a DataFrame
lg_issues_df = pd.DataFrame(
    grouped_lg_surface[["NUMBER_OF_TOILETS", "NUMBER_OF_TABLES"]].sum())
lg_issues_df.head(10)
```

		NUMBER_OF_TOILETS	NUMBER_OF_TABLES
LOCAL_GOVERNMENT_NAME	**SURFACE**		
Albany (C)	**Surfaced**	0	4
	Unsurfaced	0	5
Armadale (C)	**Unsurfaced**	0	0
Ashburton	**Surfaced**	2	7
	Unsurfaced	5	2
Augusta - Margaret River	**Surfaced**	0	0
	Unsurfaced	0	0
Beverley	**Surfaced**	0	0
	Unsurfaced	0	2
Boddington	**Surfaced**	0	3

Activity: Exploring Australian Census Data

In this activity, you will use Australian Census data about rental properties to create DataFrames with calculated totals and averages of each state.

Suggested Time:

25 minutes

Time’s Up! Let’s Review.

Instructor Demonstration

Sorting Made Easy

Sorting Made Easy

To sort a DataFrame based on the values within a column, use the `df.sort_values()` method and pass in the column name to sort by as a parameter.

The "ascending" parameter is always marked as True by default. Therefore, the `sort_values()` method will always sort from lowest to highest unless the parameter of `ascending=False` is also passed into the `sort_values()` method.

Sorting Made Easy

```
# Sorting the DataFrame based on "Meals" column
# Will sort from lowest to highest if no other parameter is passed
meals_taxes_df = taxes_df.sort_values("Meals")
meals_taxes_df.head()
```

	Town	Meals	Meals Count	Rent	Rent Count	Alcohol	Alcohol Count	Past Meals	Past Meals count	Past Rent	Past Rent Count	Past Alcohol	Past Alchohol Count
0	ADDISON	0.0	0	90173.10	12	0.0	0	0.00	0	172233.00	15	0.0	0
98	WELLS	0.0	0	0.00	0	0.0	0	0.00	0	145041.00	11	0.0	0
35	FAIRLEE	0.0	0	1833212.02	10	0.0	0	2379763.68	11	4475959.53	12	0.0	0
36	FAYSTON	0.0	0	105586.77	11	0.0	0	0.00	0	211939.30	19	0.0	0
37	FERRISBURGH	0.0	0	0.00	0	0.0	0	7025450.58	11	5829011.70	15	0.0	0

```
# To sort from highest to lowest, ascending=False must be passed in
meals_taxes_df = taxes_df.sort_values("Meals", ascending=False)
meals_taxes_df.head()
```

	Town	Meals	Meals Count	Rent	Rent Count	Alcohol	Alcohol Count	Past Meals	Past Meals count	Past Rent	Past Rent Count	Past Alcohol	Past Alchohol Count
17	BURLINGTON	74507552.54	219	18230026.80	26	18324508.20	122	1.276183e+08	236	53634054.09	44	44233463.37	129
81	SOUTH BURLINGTON	64445667.13	111	13750969.61	19	4138460.85	40	8.953598e+07	117	38211751.51	25	10313786.70	44
77	RUTLAND	38005509.10	98	1508769.29	14	2973734.52	38	4.199332e+07	98	3822279.43	14	5316214.36	38
32	ESSEX	36429036.93	91	0.00	0	2359611.62	29	4.203358e+07	104	0.00	0	4129281.23	31
12	BRATTLEBORO	33966669.55	102	4868408.74	26	2840765.10	41	4.144862e+07	100	9867296.43	27	6096085.57	42

Activity: Search for the Worst

In this activity, you will use a dataset on San Francisco Airport's utility consumption and determine which day in the dataset had the worst consumption for each utility.

Suggested Time:

20 minutes

Time’s Up! Let’s Review.

Questions?