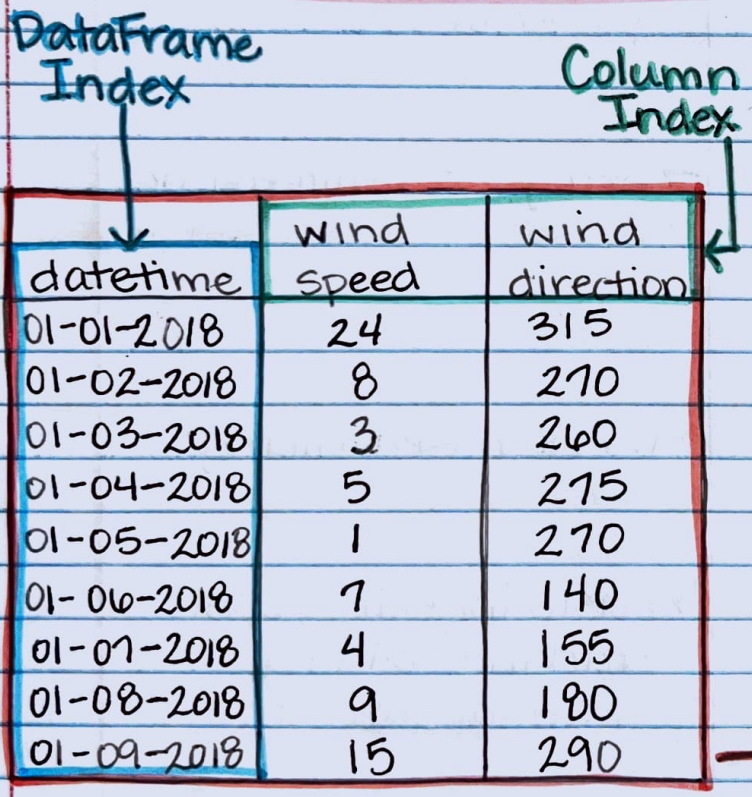
Jessica Meyer

Pandas Cheetsheet

General Purpose A powerful library used for working with large datasets, and includes functions to analyze, clean, explore, identify, and manipulate the data so it is more user-friendly. It is mostly used for data analysis tasks with a dataframe.

Pandas DataFrame A 2-dimensional data structure with corresponding labels. It includes rows and columns of potentially different data types, aligned in a tabular format, which can be accessed using labels or indexing.

Pandas DataFrame Index An object sequence that stores axis labels or identifiers for each row of data. An index can consist of hashable data types such as integers or strings, and can return the specified row data or data subset. Column labels are usually string data types, and both the dataframe index and column labels are used as unique identifiers for data within the pandas DataFrame.



DataFrame.index outputs (datetime)Index(['01-01-2023','01-02-2023',...])
DataFrame.columns outputs Index(['wind speed','wind direction'])

Pandas	vs.	Numpy
→ mostly used for large data analysis tasks		→ mostly used for working with numerical values to apply math functions
→ library works well with homogenous or heterogenous data types simultaneously. → data types: objects, integers, floats, boolean, strings, tuples		→ library works with homogenous data types only within arrays → data types: integers, floats, tuples, objects
→ DataFrames and Series are the most powerful tools		→ Arrays are the most powerful tools
→ DataFrames are 2-dimensional with headers, indices, rows, and columns.		→ Arrays are n-dimensional
→ Label-based and index-based slicing		→ index-based slicing
→ Hierarchical axis indexing, merging, joining, grouping, and subsetting.		→ basic indexing, basic stacking/splitting, minimal subsetting

Setting up a DataFrame by reading a file

After importing the pandas library, identify the file and file path.:

filename = 'streamflow_demo.txt' → also works with .csv files
filepath = **os.path.join(filename)** → if working script and data file are within the same directory

OR

filepath = os.path.join(../data_directory, filename)

OR

filepath = os.path.join(full_pathway_to_data_directory, filename)

↓ if working script and data file are within different directories

Reading data file into a pandas DataFrame and set the Index

For a .txt file → Reading datafile into pandas DataFrame

dataframe_name = **pandas.read_table(filename, how data is separated, # of rows to skip, names of the columns, parsing the dates)**

ex:

```
new_df = pandas.read_table(filename, sep='\t',
                           skiprows=31, names=['col_1', 'col_2', 'datetime'],
                           parse_dates=['datetime'])
```

For a .txt file → Setting the DataFrame Index

dataframe_name = dataframe_name.**set_index('column_name')**
↳ pandas index set function

ex:

```
new_df = new_df.set_index('datetime')
```

For a .csv file → Reading .csv data file into DataFrame

```
dataframe_name = pandas.read_csv(filename, which column to identify as index, number of rows to skip, which columns to put into the DataFrame, column names)
```

ex:

```
filename = 'streamflow_demo.csv'
new_df = pandas.read_csv(filename,
        index_col = 'datetime', skiprows = 8,
        use_cols = [1, 3, 4], names = ['col_1', 'col_2',
        'col_3', 'datetime']
```

For a .csv file → Setting the DataFrame Index to a datetime object

```
dataframe_name.index = pandas.to_datetime(
        dataframe_name.index)
```

ex:

```
new_df.index = pandas.to_datetime(new_df.index)
```

Slicing a pandas DataFrame

Using loc for rows: accessing a group of rows and/or columns by label or boolean array.

```
dataframe_name.loc[['index_A', 'index_B']] → returns a dataframe
```

or

```
dataframe_name.loc['index_A':'index_B', 'column'] → returns a series
```

or

```
dataframe_name.loc['index_A', 'column'] → returns a single object or value
```

Ex 1:

Using the prior example Pandas DataFrame = wind_df

```
wind_df.loc[['01-03-2018', '01-06-2018']]
```

output:

	wind speed	wind direction
01-03-2018	3	260
01-06-2018	7	140

→ returns a subsetted pandas DataFrame

```
wind_df.loc[[2,5]]
```

output:

	wind speed	wind direction
01-03-2018	3	260
01-06-2018	7	140

→ returns a subsetted pandas DataFrame

```
wind_df.loc['01-03-2018':'01-06-2018', 'wind speed']
```

output:

	wind speed
01-03-2018	3
01-04-2018	5
01-05-2018	1
01-06-2018	7

→ returns a subsetted series

```
wind_df.loc[2:5, 0]
```

output:

	wind speed
01-03-2018	3
01-04-2018	5
01-05-2018	1
01-06-2018	7

→ returns a subsetted series

```
wind_df.loc['01-03-2018', 'wind direction']
```

output: 260

→ returns a single object or value

Using iloc for rows : accessing a group of rows and/or columns based on integer location

`dataframe_name.iloc[[index_row]]`

output:

	col_1	col_2	col_3
index_row	value	value	value

→ returns a subsetted DataFrame

`dataframe_name.iloc[[index_rowA, index_rowB]]`

output:

	col_1	col_2	col_3
index_rowA	value	value	value
index_rowB	value	value	value

→ returns a subsetted DataFrame

`dataframe_name.iloc[index_row, index_column]`

output: value or object or string

→ returns a single value/object/string

`dataframe_name.iloc[[row_indexA, row_indexB], [column_indexC, column_indexD]]`

output:

	column_indexC	column_indexD
row_indexA	return	return
row_indexB	return	return

→ returns a subsetted DataFrame

`dataframe_name.iloc[index_rowA:index_rowC, index_columnB:index_columnC]`

output:

	Column_indexB	Column_indexC
row_indexA	return	return
row_indexB	return	return
row_indexC	return	return

Ex 2:

Using the prior example pandas DataFrame = wind_df

```
wind_df.iloc[[4]]
```

output:

	wind speed	wind direction
01-05-2018	1	270

→ returns a subsetted DataFrame

```
wind_df.iloc[[4,7]]
```

output:

	wind speed	wind direction
01-05-2018	1	270
01-08-2018	9	180

→ returns a subsetted DataFrame

```
wind_df.iloc[4,1]
```

output: 270

→ returns a single value/object/string

```
wind_df.iloc[[4,7],[1,0]]
```

output:

	wind direction	wind speed
01-05-2018	270	1
01-08-2018	180	9

→ returns a subsetted DataFrame

```
wind_df.iloc[0]
```

output:

wind speed	24
wind speed	8
wind speed	3
wind speed	5
wind speed	1
wind speed	7
wind speed	4
wind speed	9
wind speed	15

→ returns a pandas series or column without the 'datetime' index

Slicing columns by name or number

dataframe_name.loc[['column_nameA','column_nameD']]

output:

	column_namA	column_nameD
index_0	return	return
index_1	return	return
index_etc	return	return

→ returns a 2D-array, keeping column names

dataframe_name.loc['column_nameA']

output: [return 1, return2, return3]

index_0	return1
index_1	return2
index_etc	return3

→ returns a 1D-array, without column names

dataframe_name.iloc[:,[column_index0, column_index1]]

output:

	column_index0	column_index1
index_0	return	return
index_1	return	return
index_etc	return	return

Ex 3:

Using the prior example pandas DataFrame = wind_df

wind_df.loc[['wind direction']]

output:

	wind direction
01-01-2018	315
01-02-2018	270
⋮	
01-08-2018	180
01-09-2018	290

→ returns a 2D-array, keeping column names

wind_df.loc['wind direction']

output: [315, 270, 260, 275, 270, 140, 155, 180, 290]

→ returns a 1D-array, without column names

wind_df.iloc[:, [1]]

output:

	wind direction
01-01-2018	315
01-02-2018	270
⋮	
01-08-2018	180
01-09-2018	290

Helpful Pandas Functions and Methods

pandas.DataFrame_name.sort_values('column_to_sort_by', specify_ascending_or_descending, ignore_index_or_not)

→ return a dataframe that has been sorted by a specified column, either including the associated index or not during the sorting

ex: pandas.wind_df.sort_values(by='wind speed', ascending=True, ignore_index=False)

output:

	wind speed	wind direction
datetime		
01-05-2018	1	270
01-03-2018	3	260
⋮	⋮	⋮
01-09-2018	15	290
01-01-2018	24	315

if ascending=False & ignore_index=True →

Index	wind speed	wind direction
0	24	315
1	15	290
⋮	⋮	⋮
7	3	260
8	1	270

pandas.DataFrame = pandas.DataFrame.dropna(specify axis that is dropped)

→ drops any missing data or nan values, as well as the accompanying (user specified) index.

ex: wind_df = wind_df.dropna(axis=0)

→ if axis=0, rows with missing data are dropped

→ if axis=1, columns with missing data are dropped

pandas.DataFrame.columns

→ prints the column index and datatype

ex: wind_df.columns

output: Index(['wind speed', 'wind direction'], dtype='object')

pandas.DataFrame.describe()

→ prints the number of objects/values in the length of the dataframe, the mean, standard deviation, minimum, maximum, and quantiles for each column.

ex: wind_df.describe()

output:

	wind speed	wind direction
count	9	9
mean	8.44	239.45
std	value	value
min	1	140
25%	value	value
50%	value	value
75%	value	value
max	24	315

pandas.DataFrame.insert(new_column_index, 'new_column_name', pandas_Series)

→ inserts a new column into the dataframe at specified index

ex: temp_f = pandas.Series([74, 43, 40, 41, 49, 57, 59, 62, 66])

wind_df.insert(0, 'temp_f', temp_f)

output:

datetime	temp-f	wind speed	wind direction
01-01-2018	74	24	315