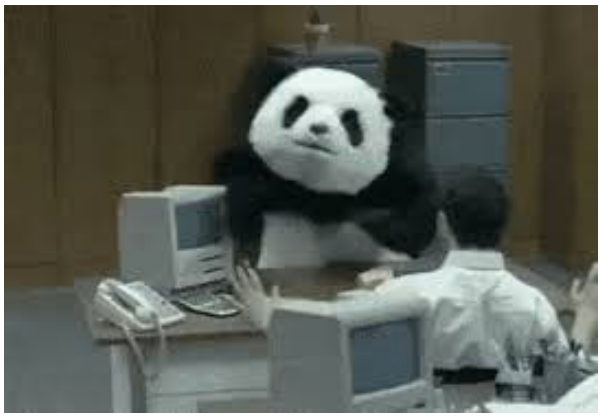
Advanced Pandas – Part 1



Dennis J. Zhang
Washington University in St. Louis

Changing Column Names

```
import pandas as pd

df_grades = pd.read_csv("Data/Grades_Short.csv")
df_grades
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A
5	Tarik	31.0	1	19.0	19	1	8.0	24.0	B
6	Malik	31.5	1	20.0	21	1	9.0	36.0	A

```
#Get column names
df_grades.columns
```

```
Index(['Name', 'Previous_Part', 'Participation1', 'Mini_Exam1', 'Mini_Exam2',
       'Participation2', 'Mini_Exam3', 'Final', 'Grade'],
      dtype='object')
```

Changing Column Names

```
#Get column names
df_grades.columns
```

```
Index(['Name', 'Previous_Part', 'Participation1', 'Mini_Exam1', 'Mini_Exam2',
       'Participation2', 'Mini_Exam3', 'Final', 'Grade'],
      dtype='object')
```

```
#Changing column names
df_grades.rename(columns={"Mini_Exam1": "Mini_Exam_1","Mini_Exam2": "Mini_Exam_2"}, \
                          inplace = False)
```

Changing Column Names


```
#Get column names
df_grades.columns
```

```
Index(['Name', 'Previous_Part', 'Participation1', 'Mini_Exam1', 'Mini_Exam2',
       'Participation2', 'Mini_Exam3', 'Final', 'Grade'],
      dtype='object')
```

```
#Changing column names
df_grades.rename(columns={"Mini_Exam1": "Mini_Exam_1","Mini_Exam2": "Mini_Exam_2"}, \
                          inplace = False)
```


"curly" brackets around new column name assignments

Changing Column Names

```
#Get column names
df_grades.columns
```

```
Index(['Name', 'Previous_Part', 'Participation1', 'Mini_Exam1', 'Mini_Exam2',
       'Participation2', 'Mini_Exam3', 'Final', 'Grade'],
      dtype='object')
```

```
#Changing column names
df_grades.rename(columns={"Mini_Exam1": "Mini_Exam_1","Mini_Exam2": "Mini_Exam_2"}, \
                          inplace = False)
```

"old_column_name": "new_column_name"

Changing Column Names

```
#Get column names
df_grades.columns
```

```
Index(['Name', 'Previous_Part', 'Participation1', 'Mini_Exam1', 'Mini_Exam2',
       'Participation2', 'Mini_Exam3', 'Final', 'Grade'],
      dtype='object')
```

```
#Changing column names
df_grades.rename(columns={"Mini_Exam1": "Mini_Exam_1","Mini_Exam2": "Mini_Exam_2"}, \
                          inplace = False)
```

inplace = False (default) returns a new dataframe
(df_grades is unaltered) with updated column names.

Changing Column Names

```
#Get column names
df_grades.columns
```

```
Index(['Name', 'Previous_Part', 'Participation1', 'Mini_Exam1', 'Mini_Exam2',
       'Participation2', 'Mini_Exam3', 'Final', 'Grade'],
      dtype='object')
```

```
#Changing column names
df_grades.rename(columns={"Mini_Exam1": "Mini_Exam_1","Mini_Exam2": "Mini_Exam_2"}, \
                                 inplace = False)
```

	Name	Previous_Part	Participation1	Mini_Exam_1	Mini_Exam_2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A
5	Tarik	31.0	1	19.0	19	1	8.0	24.0	B
6	Malik	31.5	1	20.0	21	1	9.0	36.0	A

Concatenating DataFrames - Stacked

```
df_grades_A
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A
3	Chris	30.0	1	19.0	17	1	12.5	33.5	A
4	Malik	31.5	1	20.0	21	1	9.0	36.0	A

```
df_grades_other
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
1	Tarik	31.0	1	19.0	19	1	8.0	24.0	B

Let's say you had separate csv files with the info for the students who got an A and everyone else, but you want to analyze everything together.

Concatenating DataFrames - Stacked

```
df_grades = pd.concat([df_grades_A,df_grades_other],\
                      axis = 0)
```

List of dataframes to concatenate

Concatenating DataFrames - Stacked

```
df_grades = pd.concat([df_grades_A,df_grades_other],\
                      axis = 0)
```

axis = 0 (default) – combine the two dataframes by stacking them on top of each other. Set axis =1 to stack side by side.

Concatenating DataFrames - Stacked

```
df_grades_A
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A
3	Chris	30.0	1	19.0	17	1	12.5	33.5	A
4	Malik	31.5	1	20.0	21	1	9.0	36.0	A

```
df_grades_other
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
1	Tarik	31.0	1	19.0	19	1	8.0	24.0	B

```
df_grades = pd.concat([df_grades_A,df_grades_other],\
                      axis = 0)
```

- # of columns has to match
- What is going to happen to index?

Concatenating DataFrames - Stacked

```
df_grades = pd.concat([df_grades_A,df_grades_other],\
                      axis = 0)
```

```
df_grades
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A
3	Chris	30.0	1	19.0	17	1	12.5	33.5	A
4	Malik	31.5	1	20.0	21	1	9.0	36.0	A
0	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
1	Tarik	31.0	1	19.0	19	1	8.0	24.0	B

- We got a repeated index, which is not good!
- We will eventually see how to “reset” the index.
- We can also correct this when we concatenate.

Concatenating DataFrames - Stacked

```
df_grades = pd.concat([df_grades_A,df_grades_other],\
                      axis = 0)
```

```
df_grades
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A
3	Chris	30.0	1	19.0	17	1	12.5	33.5	A
4	Malik	31.5	1	20.0	21	1	9.0	36.0	A
0	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
1	Tarik	31.0	1	19.0	19	1	8.0	24.0	B

- We got a repeated index, which is not good!
- We will eventually see how to “reset” the index.
- We can also correct this when we concatenate.

Concatenating DataFrames - Stacked

```
df_grades = pd.concat([df_grades_A,df_grades_other],\
                      axis = 0, ignore_index= True)
```

```
df_grades
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A
3	Chris	30.0	1	19.0	17	1	12.5	33.5	A
4	Malik	31.5	1	20.0	21	1	9.0	36.0	A
5	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
6	Tarik	31.0	1	19.0	19	1	8.0	24.0	B

Notice the ignore_index input!

Using the Index

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A
5	Tarik	31.0	1	19.0	19	1	8.0	24.0	B
6	Malik	31.5	1	20.0	21	1	9.0	36.0	A

- The index in this case is row numbers.
- What if I want to quickly see Joe's row?
    - I have to look up what row Joe is in.
    - Instead, I can make the index the column name.

Using the Index

```
df_grades.set_index("Name", inplace=True)
df_grades
```

Column that will become index (make sure this is unique).

Using the Index

```
df_grades.set_index("Name", inplace=True)
df_grades
```

Modify df_grades

Using the Index

```
df_grades.set_index("Name", inplace=True)
df_grades
```

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
Name								
Jake	32.0	1	19.5	20	1	10.0	33.0	A
Joe	32.0	1	20.0	16	1	14.0	32.0	A
Susan	30.0	1	19.0	19	1	10.5	33.0	A-
Sol	31.0	1	22.0	13	1	13.0	34.0	A
Chris	30.0	1	19.0	17	1	12.5	33.5	A
Tarik	31.0	1	19.0	19	1	8.0	24.0	B
Malik	31.5	1	20.0	21	1	9.0	36.0	A

The index is now the name column!

Using the Index

```
df_grades.set_index("Name", inplace=True)
df_grades
```

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
Name								
Jake	32.0	1	19.5	20	1	10.0	33.0	A
Joe	32.0	1	20.0	16	1	14.0	32.0	A
Susan	30.0	1	19.0	19	1	10.5	33.0	A-
Sol	31.0	1	22.0	13	1	13.0	34.0	A
Chris	30.0	1	19.0	17	1	12.5	33.5	A
Tarik	31.0	1	19.0	19	1	8.0	24.0	B
Malik	31.5	1	20.0	21	1	9.0	36.0	A

```
#Easy to find Joe's grade
df_grades.loc["Joe", "Grade"]
```

```
'A'
```

Using the Index

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	ID
Name									
Jake	32.0	1	19.5	20	1	10.0	33.0	A	90743
Joe	32.0	1	20.0	16	1	14.0	32.0	A	7284
Susan	30.0	1	19.0	19	1	10.5	33.0	A-	7625
Sol	31.0	1	22.0	13	1	13.0	34.0	A	1237
Chris	30.0	1	19.0	17	1	12.5	33.5	A	62
Tarik	31.0	1	19.0	19	1	8.0	24.0	B	87452
Malik	31.5	1	20.0	21	1	9.0	36.0	A	9374

Want to make new ID column the index

Using the Index

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	ID
Name									
Jake	32.0	1	19.5	20	1	10.0	33.0	A	90743
Joe	32.0	1	20.0	16	1	14.0	32.0	A	7284
Susan	30.0	1	19.0	19	1	10.5	33.0	A-	7625
Sol	31.0	1	22.0	13	1	13.0	34.0	A	1237
Chris	30.0	1	19.0	17	1	12.5	33.5	A	62
Tarik	31.0	1	19.0	19	1	8.0	24.0	B	87452
Malik	31.5	1	20.0	21	1	9.0	36.0	A	9374

```
df_grades.set_index("ID", inplace=False)
```

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
ID								
90743	32.0	1	19.5	20	1	10.0	33.0	A
7284	32.0	1	20.0	16	1	14.0	32.0	A
7625	30.0	1	19.0	19	1	10.5	33.0	A-
1237	31.0	1	22.0	13	1	13.0	34.0	A
62	30.0	1	19.0	17	1	12.5	33.5	A
87452	31.0	1	19.0	19	1	8.0	24.0	B
9374	31.5	1	20.0	21	1	9.0	36.0	A

This accomplishes the task but we lose the name column.

Using the Index

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	ID
Name									
Jake	32.0	1	19.5	20	1	10.0	33.0	A	90743
Joe	32.0	1	20.0	16	1	14.0	32.0	A	7284
Susan	30.0	1	19.0	19	1	10.5	33.0	A-	7625
Sol	31.0	1	22.0	13	1	13.0	34.0	A	1237
Chris	30.0	1	19.0	17	1	12.5	33.5	A	62
Tarik	31.0	1	19.0	19	1	8.0	24.0	B	87452
Malik	31.5	1	20.0	21	1	9.0	36.0	A	9374

```
#Add names column
df_grades["Name"] = df_grades.index
#Get new ID indes
df_grades.set_index("ID", inplace=True)
df_grades
```

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Name
ID									
90743	32.0	1	19.5	20	1	10.0	33.0	A	Jake
7284	32.0	1	20.0	16	1	14.0	32.0	A	Joe
7625	30.0	1	19.0	19	1	10.5	33.0	A-	Susan
1237	31.0	1	22.0	13	1	13.0	34.0	A	Sol
62	30.0	1	19.0	17	1	12.5	33.5	A	Chris
87452	31.0	1	19.0	19	1	8.0	24.0	B	Tarik
9374	31.5	1	20.0	21	1	9.0	36.0	A	Malik

Using the Index

	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	ID
Name									
Jake	32.0	1	19.5	20	1	10.0	33.0	A	90743
Joe	32.0	1	20.0	16	1	14.0	32.0	A	7284
Susan	30.0	1	19.0	19	1	10.5	33.0	A-	7625
Sol	31.0	1	22.0	13	1	13.0	34.0	A	1237
Chris	30.0	1	19.0	17	1	12.5	33.5	A	62
Tarik	31.0	1	19.0	19	1	8.0	24.0	B	87452
Malik	31.5	1	20.0	21	1	9.0	36.0	A	9374

```
#Add names column
df_grades["Name"] = df_grades.index
#Get new ID indes
df_grades.set_index("ID", inplace=True)
#reset the index
df_grades.reset_index(inplace=True)
df_grades
```

	ID	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Name
0	90743	32.0	1	19.5	20	1	10.0	33.0	A	Jake
1	7284	32.0	1	20.0	16	1	14.0	32.0	A	Joe
2	7625	30.0	1	19.0	19	1	10.5	33.0	A-	Susan
3	1237	31.0	1	22.0	13	1	13.0	34.0	A	Sol
4	62	30.0	1	19.0	17	1	12.5	33.5	A	Chris
5	87452	31.0	1	19.0	19	1	8.0	24.0	B	Tarik
6	9374	31.5	1	20.0	21	1	9.0	36.0	A	Malik

Using the Index

```
#Now make ID the index
df_grades.set_index("ID", inplace=True)
df_grades
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
ID									
90743	Jake	32.0	1	19.5	20	1	10.0	33.0	A
7284	Joe	32.0	1	20.0	16	1	14.0	32.0	A
7625	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
1237	Sol	31.0	1	22.0	13	1	13.0	34.0	A
62	Chris	30.0	1	19.0	17	1	12.5	33.5	A
87452	Tarik	31.0	1	19.0	19	1	8.0	24.0	B
9374	Malik	31.5	1	20.0	21	1	9.0	36.0	A

Using the Index

```
#First reset the index
df_grades.reset_index(drop=False, inplace=True)
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	ID
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	90743
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A	7284
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-	7625
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A	1237
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A	62
5	Tarik	31.0	1	19.0	19	1	8.0	24.0	B	87452
6	Malik	31.5	1	20.0	21	1	9.0	36.0	A	9374

- reset_index() will make your index row numbers again.
- Useful when manipulating dataframes and index can get messed up

Missing Data

A	B	C	D	E	F	G	H	I	J
Name	Previous_Par	Participation	Mini_Exam1	Mini_Exam2	Participation	Mini_Exam3	Final	Grade	Temp
Jake	32	1	19.5	20	1	10	33	A	-1
Joe	NA	1	20	16	1	14	32	A	23
Sol	31	1	22	13	1	13	34	A	34
Chris	30	-1	19	not available	1	12.5	33.5	A	72

```
df_missing = pd.read_csv("Data/Missing_Data.csv")
df_missing
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	-1
1	Joe	NaN	1	20.0	16	1	14.0	32.0	A	23
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A	34
3	Chris	30.0	-1	19.0	not available	1	12.5	33.5	A	72

Not that different columns have different indicators for missing data.

Missing Data

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	-1
1	Joe	NaN	1	20.0	16	1	14.0	32.0	A	23
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A	34
3	Chris	30.0	-1	19.0	not available	1	12.5	33.5	A	72

```
df_missing.dtypes
```

```
Name                object
Previous_Part      float64
Participation1       int64
Mini_Exam1         float64
Mini_Exam2          object
Participation2       int64
Mini_Exam3         float64
Final              float64
Grade               object
Temp                 int64
dtype: object
```

We can replace the missing data with a true NaN (right now everything is just a string).

Missing Data

```
df_missing = pd.read_csv("Data/Missing_Data.csv", \
                         na_values=["NaN", "not available"])
df_missing
```

List of strings specifying which values are missing.

Missing Data

```
df_missing = pd.read_csv("Data/Missing_Data.csv", \
                         na_values=["NaN", "not available"])
df_missing
```

List of strings specifying which values are missing.

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1	19.5	20.0	1	10.0	33.0	A	-1
1	Joe	NaN	1	20.0	16.0	1	14.0	32.0	A	23
2	Sol	31.0	1	22.0	13.0	1	13.0	34.0	A	34
3	Chris	30.0	-1	19.0	NaN	1	12.5	33.5	A	72

Special NaN value (from numpy package), which is not a string.

Missing Data

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	-1
1	Joe	NaN	1	20.0	16	1	14.0	32.0	A	23
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A	34
3	Chris	30.0	-1	19.0	not available	1	12.5	33.5	A	72

We know “NaN” and “not available” are missing data points, but what about -1?

Missing Data

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	-1
1	Joe	NaN	1	20.0	16	1	14.0	32.0	A	23
2	Sol	31.0	1	22.0	13	1	13.0	34.0	A	34
3	Chris	30.0	-1	19.0	not available	1	12.5	33.5	A	72

We know “NaN” and “not available” are missing data points, but what about -1?

- For the Participation1 column the -1 is probably missing data.
- For the Temp column, the -1 is likely not missing data, since -1 is a valid temperature.

For each column, we can specify exactly which values correspond to missing data.

Missing Data

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values={"Mini_Exam2" : "not available",\
                                                             "Participation1": -1})
df_missing
```

Curly brackets

Missing Data

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values={"Mini_Exam2" : "not available",\
                                                             "Participation1": -1})
df_missing
```

Column name as string

NaN value

Missing Data

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values={"Mini_Exam2" : "not available",\
                                                             "Participation1": -1})
df_missing
```

Column name as string

NaN value

"For column Participation1, replace all -1s with a NaN."

Missing Data

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values={"Mini_Exam2" : "not available",\
                                                              "Participation1": -1})
df_missing
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	-1
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72

Notice that the -1 was replaced only in Participation1 column

Comparing Approaches

Approach 1:

- Does a global search and replace in all columns.

```
df_missing = pd.read_csv("Data/Missing_Data.csv", \
                         na_values=["NaN", "not available"])
df_missing
```

Approach 2:

- Allows you to specify column by column the values that should be replaced with NaN.

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values={"Mini_Exam2" : "not available",\
                                                              "Participation1": -1})
df_missing
```

Benefiting of Having NaNs

- Have common symbol for where there is missing data
  - Good for you and good for others looking at your code/data
  - These entries will be ignored if you try to compute means of columns with NaNs.
- We can easily get rid of column/rows with missing data
- We can easily replace the missing values with the mean of the column, for example.

Isnull() Method

- The isnull() method lets you check where the NaNs are:

```
df = pd.read_csv("Data/Missing_Data.csv", na_values=["NaN", -1, "not available"])
df
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	NaN
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72.0

```
#Using isnull()
df.isnull()
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	False	False	False	False	False	False	False	False	False	True
1	False	True	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	True	False	True	False	False	False	False	False

Isnull() Method

- The isnull() method lets you check where the NaNs are:

```
#Using isnull()
df.isnull()
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	False	False	False	False	False	False	False	False	False	True
1	False	True	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	True	False	True	False	False	False	False	False

```
#Remember Booleans are just 0s and 1s.
#Check how many NaNs are in each column
df.isnull().sum()
```

```
Name               0
Previous_Part      1
Participation1     1
Mini_Exam1         0
Mini_Exam2         1
Participation2     0
Mini_Exam3         0
Final              0
Grade              0
Temp               1
dtype: int64
```

Dropna() Method

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values=["NaN", "not available",\
                                                             -1])
df_missing
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	NaN
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72.0

How do I get rid of all rows with NaN?

Dropna() Method

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values=["NaN", "not available",\
                                                              -1])
df_missing
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	NaN
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72.0

How do I get rid of all rows with NaN?

```
df_missing.dropna(axis = 0, inplace=False)
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0

- Setting axis = 1 would drop all columns with an NaN

Fillna() Method

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values=["NaN", "not available",\
                                                              -1])
df_missing
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	NaN
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72.0

Rather than getting rid of rows/columns, we fill the “holes” in a number of ways.

```
#Replace with specific value
df_missing.fillna(0, inplace=False)
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	0.0
1	Joe	0.0	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	0.0	19.0	0.0	1	12.5	33.5	A	72.0

Fillna() Method

```
df_missing = pd.read_csv("Data/Missing_Data.csv", na_values=["NaN", "not available",\
                                                              -1])
df_missing
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	NaN
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72.0

Rather than getting rid of rows/columns, we fill the “holes” in a number of ways.

```
#Replace with specific value in specific column
mean_temp = df_missing.Temp.mean()
df_missing.fillna({'Temp': mean_temp}, inplace=False)
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	Temp
0	Jake	32.0	1.0	19.5	20.0	1	10.0	33.0	A	43.0
1	Joe	NaN	1.0	20.0	16.0	1	14.0	32.0	A	23.0
2	Sol	31.0	1.0	22.0	13.0	1	13.0	34.0	A	34.0
3	Chris	30.0	NaN	19.0	NaN	1	12.5	33.5	A	72.0

Advanced Pandas – Part 2

Advanced Pandas

- Use string methods to create and manipulate columns
- Apply to create new columns that are complex functions of other columns.
- Iterating through the rows of a dataframe

Titanic Data Set

Passenger data from people onboard the Titanic:

```
df_titanic.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
df_titanic.dtypes
```

```
PassengerId      int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Using String Methods

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
df_titanic["Name"].str.lower()
```

After column name you need .str if you wan to use a string method

Using String Methods

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
df_titanic["Name"].str.lower()
```

After column name you need .str if you wan to use a string method

.“string_method”

Using String Methods

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
df_titanic["Name"].str.lower()
```

```
0                                kelly, mr. james
1                wilkes, mrs. james (ellen needs)
2                       myles, mr. thomas francis
3                                wirz, mr. albert
4    hirvonen, mrs. alexander (helga e lindqvist)
Name: Name, dtype: object
```

Returns series with all lower case in the name column.

Using String Methods

Let's say I wanted to know the proportion of "Mrs." in the data set

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
df_titanic["Name"].str.contains("Mrs")
```

```
0    False
1     True
2    False
3    False
4     True
Name: Name, dtype: bool
```

Using String Methods

Let's say I wanted to know the proportion of "Mrs." in the data set

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
df_titanic["Name"].str.contains("Mrs")
```

```
0    False
1     True
2    False
3    False
4     True
Name: Name, dtype: bool
```

Returns boolean series with True if name contains "Mrs." and False otherwise.

Using String Methods

Let's say I wanted to know the proportion of "Mrs." in the data set

```
df_titanic["Bool_Mrs"] = df_titanic["Name"].str.contains("Mrs")
```

Create new column column called "Bool_Mrs" that stores these booleans:

```
df_titanic
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Bool_Mrs
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	False
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	True
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	False
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	False
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	True

New column

Using String Methods

Let's say I wanted to know the proportion of "Mrs." in the data set

```
df_titanic["Bool_Mrs"] = df_titanic["Name"].str.contains("Mrs")
```

Create new column column called "Bool_Mrs" that stores these booleans:

```
df_titanic
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Bool_Mrs
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	False
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	True
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	False
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	False
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	True

```
#Get fraction of Mrs
df_titanic["Bool_Mrs"].mean()
```

```
0.40000000000000002
```

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

The apply() method lets us use custom or built in functions to multiple columns.

Let's begin by applying built in functions…

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

```
df_numeric.apply(max, axis  = 0)
```

Name of function

Axis = 0 - apply column wise
Axis = 1 - apply row wise

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

```
df_numeric.apply(max, axis  = 0)
```

```
Age      62.0000
Fare     12.2875
dtype: float64
```

Returns series where the index is the column name.

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

```
df_numeric.apply(max, axis  = 1)
```

```
0    34.5
1    47.0
2    62.0
3    27.0
4    22.0
dtype: float64
```

Changed axis!

Returns series where the index is the index of the dataframe.

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

```
df_numeric.apply(max, axis  = 1)
```

```
0    34.5
1    47.0
2    62.0
3    27.0
4    22.0
dtype: float64
```

Changed axis!

Returns series where the index is the index of the dataframe.

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

```
import numpy as np

df_numeric.Fare.apply(np.ceil)
```

Applying numpy ceiling function to the column fare

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric = df_titanic[["Age", "Fare"]]
df_numeric
```

	Age	Fare
0	34.5	7.8292
1	47.0	7.0000
2	62.0	9.6875
3	27.0	8.6625
4	22.0	12.2875

```
import numpy as np

df_numeric.Fare.apply(np.ceil)
```

```
0      8.0
1      7.0
2     10.0
3      9.0
4     13.0
Name: Fare, dtype: float64
```

We can use apply to create new columns!

Using Apply

Let's only pick out a couple of the numerical columns.

```
df_numeric["Ceil_Fare"] = df_numeric.Fare.apply(np.ceil)
df_numeric
```

```
/Users/feldman/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  if __name__ == '__main__':
```

Pesky SettingWithCopyWarning

	Age	Fare	Ceil_Fare
0	34.5	7.8292	8.0
1	47.0	7.0000	7.0
2	62.0	9.6875	10.0
3	27.0	8.6625	9.0
4	22.0	12.2875	13.0

New column

SettingWithCopyWarning

The way to avoid this warning:

set df_numeric equal to a copy of the slice of the dataframe

```
df_numeric = df_titanic[["Age", "Fare"]].copy()

df_numeric["Ceil_Fare"] = df_numeric.Fare.apply(np.ceil)

df_numeric
```

	Age	Fare	Ceil_Fare
0	34.5	7.8292	8.0
1	47.0	7.0000	7.0
2	62.0	9.6875	10.0
3	27.0	8.6625	9.0
4	22.0	12.2875	13.0

Using Apply

Let's see how we can use apply with a custom function on a single column:

```
df_age_sex = df_titanic[["Age", "Sex" ]].copy()

df_age_sex
```

	Age	Sex
0	34.5	male
1	47.0	female
2	62.0	male
3	27.0	male
4	22.0	female

Let's use apply to create a binary column that is 1 if the passenger is male and 0 if female.

Using Apply

Let's see how we can use apply with a custom function on a single column:

```
def Get_Gender_Bool(gender):
    print(gender)
    if gender == "male":
        return 1
    else:
        return 0
```

```
df_age_sex["Bool_Male"] = df_age_sex.Sex.apply(Get_Gender_Bool)
```

```
male
female
male
male
female
```

Using Apply

Let's see how we can use apply with a custom function on a single column:

```
def Get_Gender_Bool(gender):

    if gender == "male":
        return 1
    else:
        return 0
```

```
df_age_sex["Bool_Male"] = df_age_sex.Sex.apply(Get_Gender_Bool)

df_age_sex
```

	Age	Sex	Bool_Male
0	34.5	male	1
1	47.0	female	0
2	62.0	male	1
3	27.0	male	1
4	22.0	female	0

Calls Get_Gender_Bool once for each row.

Using Apply

Let's see how we can use apply with a custom function on multiple columns:

```
df_age_sex = df_titanic[["Age", "Sex" ]].copy()

df_age_sex
```

	Age	Sex
0	34.5	male
1	47.0	female
2	62.0	male
3	27.0	male
4	22.0	female

Let's use apply to create a binary column that is the passenger is above 60 years old and 0 otherwise.

Using Apply

Let's see how we can use apply with a custom function on a single column:

```
def Get_Senior_Bool(age):
    print(age)
    if age >=60:
        return 1
    else:
        return 0
```

```
df_age_sex["Bool_Senior"] = df_age_sex.Age.apply(Get_Senior_Bool)
```

```
34.5
47.0
62.0
27.0
22.0
```

Using Apply

Let's see how we can use apply with a custom function on a single column:

```
def Get_Senior_Bool(age):

    if age >=60:
        return 1
    else:
        return 0
```

```
df_age_sex["Bool_Senior"] = df_age_sex.Age.apply(Get_Senior_Bool)
df_age_sex
```

	Age	Sex	Bool_Senior
0	34.5	male	0
1	47.0	female	0
2	62.0	male	1
3	27.0	male	0
4	22.0	female	0

I'll apply my custom function on the age column.

Using Apply

Last one:

```
df_name_age = df_titanic[["Name", "Age" ]].copy()

df_name_age
```

	Name	Age
0	Kelly, Mr. James	34.5
1	Wilkes, Mrs. James (Ellen Needs)	47.0
2	Myles, Mr. Thomas Francis	62.0
3	Wirz, Mr. Albert	27.0
4	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	22.0

Let's use apply to a new column called Title that's stores each passenger's title (Mr./Mrs.)

Using Apply

Last one:

```
def Get_Title(name):

    parsed_name = name.split(" ")
    title = parsed_name[1]

    return title
```

```
df_name_age["Title"] = df_name_age.Name.apply(Get_Title)
df_name_age
```

	Name	Age	Title
0	Kelly, Mr. James	34.5	Mr.
1	Wilkes, Mrs. James (Ellen Needs)	47.0	Mrs.
2	Myles, Mr. Thomas Francis	62.0	Mr.
3	Wirz, Mr. Albert	27.0	Mr.
4	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	22.0	Mrs.

Using Apply

Let's see how we can use apply with a custom function on a data frame:

```
df_age_sex = df_titanic[["Age", "Sex" ]].copy()

df_age_sex
```

	Age	Sex
0	34.5	male
1	47.0	female
2	62.0	male
3	27.0	male
4	22.0	female

Let's use apply to create a binary column is 1 if the passenger is male and over 60 and 0 otherwise.

Using Apply

Let's see how we can use apply with a custom function on a data frame:

```
def Old_Man(row):

    age = row["Age"]
    sex = row["Sex"]

    if age >=60 and sex == "male":
        return 1
    else:
        return 0
```

```
df_age_sex["Old_Man"] = df_age_sex.apply(Old_Man, axis =1)
df_age_sex
```

	Age	Sex	Old_Man
0	34.5	male	0
1	47.0	female	0
2	62.0	male	1
3	27.0	male	0
4	22.0	female	0

Each row gets passed as a series to the function, where the column names are the index.

Using Apply

Let's see how we can use apply with a custom function on a data frame:

```
def Old_Man(row):

    print("-------")
    print(row)
    print("-------")
    age = row["Age"]
    sex = row["Sex"]

    if age >=60 and sex == "male":
        return 1
    else:
        return 0
```

```
df_age_sex["Old_Man"] = df_age_sex.apply(Old_Man, axis =1)
```

```
-------
Age    34.5
Sex    male
Name: 0, dtype: object
-------
-------
Age        47
Sex    female
Name: 1, dtype: object
-------
-------
Age      62
Sex    male
Name: 2, dtype: object
-------
-------
Age      27
Sex    male
Name: 3, dtype: object
-------
-------
Age        22
Sex    female
Name: 4, dtype: object
-------
```

The function Old_Man gets called for each row in the dataframe.

Iterrows

```
df = df_titanic[["Name", "Sex", "Age"]].copy()
df
```

	Name	Sex	Age
0	Kelly, Mr. James	male	34.5
1	Wilkes, Mrs. James (Ellen Needs)	female	47.0
2	Myles, Mr. Thomas Francis	male	62.0
3	Wirz, Mr. Albert	male	27.0
4	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0

You can use the iterrows() method to go through the rows of a dataframe using a for loop.

Iterrows

Returns two loop variables

Index of df

Row as series

```
for index, row in df.iterrows():

    print("------")
    print(index)
    print(row)
    print("------")
```

Iterrows

```
for index, row in df.iterrows():

    print("------")
    print(index)
    print(row)
    print("------")
```

```
------
0
Name    Kelly, Mr. James
Sex                 male
Age                 34.5
Name: 0, dtype: object
------
------
1
Name    Wilkes, Mrs. James (Ellen Needs)
Sex                               female
Age                                   47
Name: 1, dtype: object
------
------
```

Iterrows

```
df = df_titanic[["Name", "Sex", "Age"]].copy()
df
```

	Name	Sex	Age
0	Kelly, Mr. James	male	34.5
1	Wilkes, Mrs. James (Ellen Needs)	female	47.0
2	Myles, Mr. Thomas Francis	male	62.0
3	Wirz, Mr. Albert	male	27.0
4	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0

Let's say I want to create a dictionary where I have an entry for each passenger where the key is the last name and the value is the age.

Iterrows

	Name	Sex	Age
0	Kelly, Mr. James	male	34.5
1	Wilkes, Mrs. James (Ellen Needs)	female	47.0
2	Myles, Mr. Thomas Francis	male	62.0
3	Wirz, Mr. Albert	male	27.0
4	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0


```
D = {}

for index, row in df.iterrows():

    name = row["Name"]
    age = row["Age"]
    last_name = name.replace(",","").split(" ")[0]
    D[last_name] = age
D
```

```
{'Hirvonen': 22.0, 'Kelly': 34.5, 'Myles': 62.0, 'Wilkes': 47.0, 'Wirz': 27.0}
```