Introduction to Data Analytics

ITC 5201

Instructor: Parisa Pouladzadeh
Email: parisa.pouladzadeh@humber.ca

Pandas

We will use pandas to:

- Read in data from Excel.
- Manipulate data in spreadsheet.

Reading in Data From Excel

I have the following data saved in the file “Grades_Short.csv”:

F30

	A	B	C	D	E	F	G	H	I
1	Name	Previous_Par	Participation	Mini_Exam1	Mini_Exam2	Participation	Mini_Exam3	Final	Grade
2	Jake	32	1	19.5	20	1	10	33	A
3	Joe	32	1	20	16	1	14	32	A
4	Susan	30	1	19	19	1	10.5	33	A-
5	Sol	31	1	22	13	1	13	34	A
6	Chris	30	1	19	17	1	12.5	33.5	A
7	Tarik	31	1	19	19	1	8	24	B
8	Malik	31.5	1	20	21	1	9	36	A
9									
10									

Let’s see how we read this data into pandas:

Reading in Data From Excel

I have the following data saved in the file “Grades_Short.csv”:

F30

	A	B	C	D	E	F	G	H	I
1	Name	Previous_Par	Participation	Mini_Exam1	Mini_Exam2	Participation	Mini_Exam3	Final	Grade
2	Jake	32	1	19.5	20	1	10	33	A
3	Joe	32	1	20	16	1	14	32	A
4	Susan	30	1	19	19	1	10.5	33	A-
5	Sol	31	1	22	13	1	13	34	A
6	Chris	30	1	19	17	1	12.5	33.5	A
7	Tarik	31	1	19	19	1	8	24	B
8	Malik	31.5	1	20	21	1	9	36	A
9									
10									

Let’s see how we read this data into pandas:

Reading the data into a variable called df_grades.

```
import pandas as pd

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

Built in read_csv method

Path to file

The head() Method

Using the **head()** method

```
import pandas as pd

df_grades = pd.read_csv("Grades_Short.csv")
df_grades.head(3)
```

	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-

- If the data is really large you don't want to print out the entire dataframe to your output.
- The **head(n)** method outputs the first n rows of the data frame. If n is not supplied, the default is the first 5 rows.
- I like to run the head() method after I read in the dataframe to check that everything got read in correctly.
- There is also a **tail(n)** method that returns the last n rows of the dataframe

Basic Features

```
import pandas as pd

df_grades = pd.read_csv("Grades_Short.csv")
df_grades.head(3)
```

	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-

```
#dimension of df
df_grades.shape
```

```
(7, 9)
```

Think of this as a list

```
#How each column is stored
df_grades.dtypes
```

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

object = string

float64 = decimal

int64 = integer

Basic Features

column names

	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-

row names = index

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

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

```
#Get row names
df_grades.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

Selecting a Single Column

	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 Name column
df_grades['Name']
```

```
0      Jake
1       Joe
2     Susan
3       Sol
4     Chris
5     Tarik
6     Malik
Name: Name, dtype: object
```

- Between square brackets, the column must be given as a string
- Outputs column as a series
  - A series is a one dimensional dataframe..more on this in the slicing section

Selecting a Single Column

	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 Name column
df_grades.Name
```

```
0     Jake
1      Joe
2    Susan
3      Sol
4    Chris
5    Tarik
6    Malik
Name: Name, dtype: object
```

- Exactly equivalent way to get Name column
  - + : don't have to type brackets or quotes
  - -: won't generalize to selecting multiple columns,, won't work if column names have spaces, can't create new columns this way

Selecting Multiple Columns

	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

```
#Select multiple columns
df_grades[["Name", "Grade"]]
```

	Name	Grade
0	Jake	A
1	Joe	A
2	Susan	A-
3	Sol	A
4	Chris	A
5	Tarik	B
6	Malik	A

- List of strings, which correspond to column names.
- You can select as many column as you want.
- Column don't have to be contiguous.

Storing Result

```
#Print the column
df_grades["Name"]
```

```
0     Jake
1      Joe
2    Susan
3      Sol
4    Chris
5    Tarik
6    Malik
Name: Name, dtype: object
```

```
#Store the column
names= df_grades["Name"]
names
```

```
0     Jake
1      Joe
2    Susan
3      Sol
4    Chris
5    Tarik
6    Malik
Name: Name, dtype: object
```

The variable name stores a series

Why store a slice?

- We might want/have to do our analysis is steps.
  - Less error prone
  - More readable

Slicing a Series

Slice/index through the index, which is usually numbers

```
names= df_grades["Name"]
names
```

```
0     Jake
1      Joe
2    Susan
3      Sol
4    Chris
5    Tarik
6    Malik
Name: Name, dtype: object
```

Slicing a Series

Slice/index through the index, which is usually numbers

```
names= df_grades["Name"]
names
```

```
0     Jake
1      Joe
2    Susan
3      Sol
4    Chris
5    Tarik
6    Malik
Name: Name, dtype: object
```

Picking out single element

```
names[0]
```

```
'Jake'
```

Slicing a Series

```
names= df_grades["Name"]
names
```

```
0      Jake
1       Joe
2     Susan
3       Sol
4     Chris
5     Tarik
6     Malik
Name: Name, dtype: object
```

Slice/index through the index, which is usually numbers

Picking out single element

```
names[0]
```

```
'Jake'
```

Contiguous slice

non_inclusive

```
names[1:4]
```

```
1       Joe
2     Susan
3       Sol
Name: Name, dtype: object
```

Slicing a Series

Slice/index through the index, which is usually numbers

```
names= df_grades["Name"]
names
```

```
0     Jake
1      Joe
2    Susan
3      Sol
4    Chris
5    Tarik
6    Malik
Name: Name, dtype: object
```

Picking out single element

```
names[0]
```

```
'Jake'
```

Contiguous slice

```
names[1:4]
```

```
1      Joe
2    Susan
3      Sol
Name: Name, dtype: object
```

Arbitrary slice

```
names[[1,2,4]]
```

```
1      Joe
2    Susan
4    Chris
Name: Name, dtype: object
```

Slicing a Data Frame

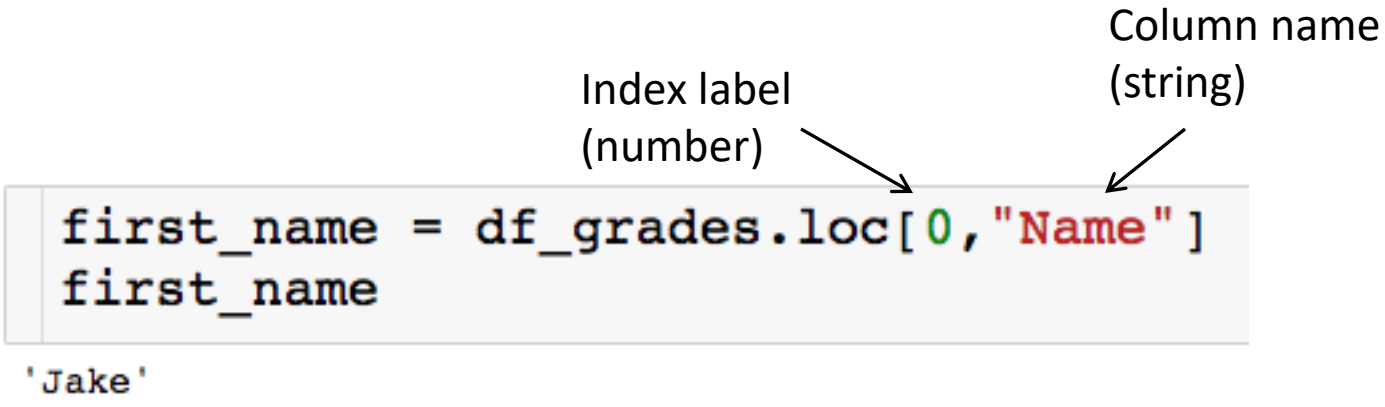
	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A
5	Tarik	31.0	1	19.0	19	1	8.0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

- There are a few ways to pick slice a data frame, we will use the .loc method.
- Access elements through the index labels column names
    - We will see how to change both of these labels later on

Slicing a Data Frame

	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

- Pick a single value out.



Slicing a Data Frame

	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

- Pick out entire row:

```
first_row = df_grades.loc[0,:]
first_row
```

```
Name              Jake
Previous_Part       32
Participation1       1
Mini_Exam1        19.5
Mini_Exam2          20
Participation2       1
Mini_Exam3          10
Final               33
Grade                A
Name: 0, dtype: object
```

"pick out all columns"

first_row is a series

Slicing a Data Frame

	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

- Pick out contiguous chunk:

Endpoints are inclusive!

```
slice_one = df_grades.loc[0:2,"Name":"Mini_Exam2"]
slice_one
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2
0	Jake	32.0	1	19.5	20
1	Joe	32.0	1	20.0	16
2	Susan	30.0	1	19.0	19

Slicing a Data Frame

	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

- Pick out arbitrary chunk:

```
slice_two = df_grades.loc[[0,2,3], ["Name", "Grade"]]
slice_two
```

	Name	Grade
0	Jake	A
2	Susan	A-
3	Sol	A

Built in Functions

	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

How do I compute the average score on the final?

```
#Print out
df_grades.Final.mean()
```

```
32.214285714285715
```

```
#Store
avg_final = df_grades.Final.mean()
avg_final
```

```
32.214285714285715
```

Built in mean() method

Built in Functions

	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

How do I compute the highest Mini Exam 1 score?

```
max_mini_1 = df_grades["Mini_Exam1"].max()
max_mini_1
```

```
22.0
```

Creating New Columns

```
import pandas as pd

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

	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~~ → $33/36$
1	Joe	32.0	1	20.0	16	1	14.0	32.0	~~A~~ → $32/36$
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

We can also create column as function of other column. The Final was worth 36 points, let's create a column for each student's percentage.

```
df_grades["Final_Percentage"]=df_grades["Final"]/36
df_grades.head()
```

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	new_column	Final_Percentage
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	1	0.916667
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A	1	0.888889
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-	1	0.916667
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A	1	0.944444
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A	1	0.930556

Deleting Columns

	Name	Previous_Part	Participation1	Mini_Exam1	Mini_Exam2	Participation2	Mini_Exam3	Final	Grade	new_column	Part_perc
0	Jake	32.0	1	19.5	20	1	10.0	33.0	A	1	1.0
1	Joe	32.0	1	20.0	16	1	14.0	32.0	A	1	1.0
2	Susan	30.0	1	19.0	19	1	10.5	33.0	A-	1	1.0
3	Sol	31.0	1	22.0	13	1	13.0	34.0	A	1	1.0
4	Chris	30.0	1	19.0	17	1	12.5	33.5	A	1	1.0

```
#Delete multiple columns
df_grades.drop(["new_column","Part_perc"], axis=1, inplace = True)
df_grades.head()
```

	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

The Drop Method

- List of column of index label

- inplace = True– change df_grades
- inplace = False – return dataframe with specified columns deleted, do not change df_grades

```
df_grades.drop(["new_column","Part_perc"], axis=1, inplace = True)
df_grades.head()
```

- axis = 1 – delete specified columns
- axis = 0 – delete specified rows

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

```
df_grades.rename?
```

```
Signature: df_grades.rename(mapper=None, index=None, columns=None, axis=None, copy=True, inplace=False, level=None)
Docstring:
Alter axes labels.

Function / dict values must be unique (1-to-1). Labels not contained in
a dict / Series will be left as-is. Extra labels listed don't throw an
error.

See the :ref:`user guide <basics.rename>` for more.

Parameters
----------
mapper, index, columns : dict-like or function, optional
    dict-like or functions transformations to apply to
    that axis' values. Use either ``mapper`` and ``axis`` to
    specify the axis to target with ``mapper``, or ``index`` and
    ``columns``.
axis : int or str, optional
    Axis to target with ``mapper``. Can be either the axis name
    ('index', 'columns') or number (0, 1). The default is 'index'.
copy : boolean, default True
    Also copy underlying data
inplace : boolean, default False
    Whether to return a new %(klass)s. If True then value of copy is
    ignored.
level : int or level name, default None
    In case of a MultiIndex, only rename labels in the specified
    level.
```

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)
```

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, 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!

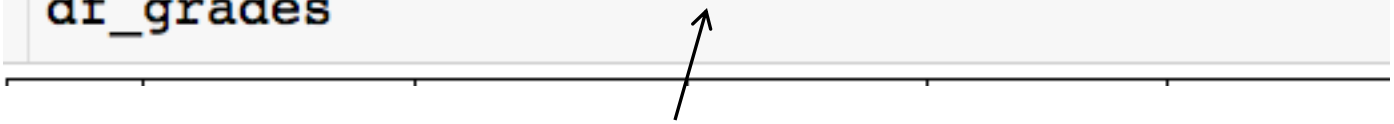
Using the Index

	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).

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).

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

Dropna() Method

```
df_missing.dropna?
```

```
Signature: df_missing.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
Docstring:
Return object with labels on given axis omitted where alternately any
or all of the data are missing

Parameters
----------
axis : {0 or 'index', 1 or 'columns'}, or tuple/list thereof
    Pass tuple or list to drop on multiple axes
how : {'any', 'all'}
    * any : if any NA values are present, drop that label
    * all : if all values are NA, drop that label
thresh : int, default None
    int value : require that many non-NA values
subset : array-like
    Labels along other axis to consider, e.g. if you are dropping rows
    these would be a list of columns to include
inplace : boolean, default False
    If True, do operation inplace and return None.
```

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