Data Science
Data Engineering I

Working with data

PURDUE UNIVERSITY | College of Science

Data wrangling

Loosely refers to the tasks needed to preprocess data, including:

- Augmentation (e.g., adding features/rows)
- Subsetting (e.g., filtering and selecting)
- Cleaning (e.g., find missing values/errors to correct)
- Aggregating (e.g., grouping by values and summarizing)
- Transforming (e.g., scaling attribute values)

Augmenting: Adding columns

```
# add a column to a data frame with a constant value
data['Tmp'] = 'testing'
```

	Index	Year	Age	Name	Movie	Tmp
0	1	1928	22	Janet Gaynor	Seventh Heaven,Street Angel and Sunrise: A Son...	testing
1	2	1929	37	Mary Pickford	Coquette	testing
2	3	1930	28	Norma Shearer	The Divorcee	testing
3	4	1931	63	Marie Dressler	Min and Bill	testing
4	5	1932	32	Helen Hayes	The Sin of Madelon Claudet	testing

```
# add a column with a value calculated from other column
data['Tmp2']=data.Year+1
data.head()
```

	Index	Year	Age	Name	Movie	Tmp	Tmp2
0	1	1928	22	Janet Gaynor	Seventh Heaven,Street Angel and Sunrise: A Son...	testing	1929
1	2	1929	37	Mary Pickford	Coquette	testing	1930
2	3	1930	28	Norma Shearer	The Divorcee	testing	1931
3	4	1931	63	Marie Dressler	Min and Bill	testing	1932
4	5	1932	32	Helen Hayes	The Sin of Madelon Claudet	testing	1933

Augmenting: Adding rows

```
# construct new example as a Series object (labeled array able to hold any type of data)
newData = [89, 2017, 29, 'Emma Stone', 'La La Land']
newExample = pd.Series(newData, index=list(data.columns))
print(newExample)
Index            89
Year           2017
Age              29
Name     Emma Stone
Movie    La La Land
dtype: object
```

```
# add rows by appending Data Frame or Data Series
# returns new object
data2 = data.append(newExample, ignore_index=True)
data2.tail()
```

	Index	Year	Age	Name	Movie
85	86	2013	22	Jennifer Lawrence	Silver Linings Playbook
86	87	2014	44	Cate Blanchett	Blue Jasmine
87	88	2015	54	Julianne Moore	Still Alice
88	89	2016	26	Brie Larson	Room
89	89	2017	29	Emma Stone	La La Land

Subsetting: Selecting columns by name

```
# select a subset of columns by name
data[['Age','Movie']]
```

	Age	Movie
0	22	Seventh Heaven,Street Angel and Sunrise: A Son...
1	37	Coquette
2	28	The Divorcee
3	63	Min and Bill
4	32	The Sin of Madelon Claudet

```
# another way to select columns by name
data.filter(items=['Year', 'Name'])
```

	Year	Name
0	1928	Janet Gaynor
1	1929	Mary Pickford
2	1930	Norma Shearer
3	1931	Marie Dressler
4	1932	Helen Hayes

Subsetting: Filtering by condition

```
# select rows that match condition in []
data[data.Year > 2010]
```

	Index	Year	Age	Name	Movie
83	84	2011	29	Natalie Portman	Black Swan
84	85	2012	62	Meryl Streep	The Iron Lady
85	86	2013	22	Jennifer Lawrence	Silver Linings Playbook
86	87	2014	44	Cate Blanchett	Blue Jasmine
87	88	2015	54	Julianne Moore	Still Alice
88	89	2016	26	Brie Larson	Room

Subsetting: Filtering by condition

```
# select rows that match condition in []
data[(data.Year > 2010) & (data.Age < 30)]
```

	Index	Year	Age	Name	Movie
83	84	2011	29	Natalie Portman	Black Swan
85	86	2013	22	Jennifer Lawrence	Silver Linings Playbook
88	89	2016	26	Brie Larson	Room

```
data[(data.Age < 25) | (data.Age > 70)]
```

	Index	Year	Age	Name	Movie
0	1	1928	22	Janet Gaynor	Seventh Heaven,Street Angel and Sunrise: A Son...
54	55	1982	74	Katharine Hepburn	On Golden Pond
59	60	1987	21	Marlee Matlin	Children of a Lesser God
62	63	1990	80	Jessica Tandy	Driving Miss Daisy
85	86	2013	22	Jennifer Lawrence	Silver Linings Playbook

Subsetting: Filtering by condition

```
# make new attribute that is True for all movies in even years
data['IsOddYr'] = data.Year%2==1
data.head()
```

	Index	Year	Age	Name	Movie	IsOddYr
0	1	1928	22	Janet Gaynor	Seventh Heaven,Street Angel and Sunrise: A Son...	False
1	2	1929	37	Mary Pickford	Coquette	True
2	3	1930	28	Norma Shearer	The Divorcee	False
3	4	1931	63	Marie Dressler	Min and Bill	True
4	5	1932	32	Helen Hayes	The Sin of Madelon Claudet	False

```
# select based on Boolean conditions
data[(data.IsOddYr) & ((data.Age==29)|(data.Age==42))]
```

	Index	Year	Age	Name	Movie	IsOddYr
47	48	1975	42	Ellen Burstyn	Alice Doesn't Live Here Anymore	True
63	64	1991	42	Kathy Bates	Misery	True
83	84	2011	29	Natalie Portman	Black Swan	True

Subsetting: Selecting rows

```
# get a random sample of ten rows
data.sample(10)
```

	Index	Year	Age	Name	Movie	IsOddYr
29	30	1957	28	Joanne Woodward	The Three Faces of Eve	True
24	25	1952	54	Shirley Booth	Come Back,Little Sheba	False
61	62	1989	26	Jodie Foster	The Accused	True
46	47	1974	37	Glenda Jackson	A Touch of Class	False
37	38	1965	25	Julie Christie	Darling	True
28	29	1956	41	Ingrid Bergman	Anastasia	False
65	66	1993	33	Emma Thompson	Howards End	True
49	50	1977	38	Faye Dunaway	Network	True
4	5	1932	32	Helen Hayes	The Sin of Madelon Claudet	False
38	39	1966	35	Elizabeth Taylor	Who's Afraid of Virginia Woolf?	False

Cleaning: Find missing values

```
# read in grades data file
data2 = pd.read_csv("grades.csv")
```

```
# find missing values via non-null counts
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 9 columns):
Lastname     16 non-null object
Firstname    16 non-null object
SSN          16 non-null object
Test1        16 non-null float64
Test2        13 non-null float64
Test3        15 non-null float64
Test4        15 non-null float64
Final        16 non-null float64
Grade        16 non-null object
dtypes: float64(5), object(4)
memory usage: 1.2+ KB
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade
0	Alfalfa	Aloysius	123-45-6789	40.0	90.0	100.0	83.0	49.0	D-
1	Alfred	University	123-12-1234	41.0	97.0	96.0	97.0	48.0	D+
2	Gerty	Gramma	567-89-0123	41.0	80.0	60.0	40.0	44.0	C
3	Android	Electric	087-65-4321	42.0	23.0	36.0	45.0	47.0	B-
4	Bumpkin	Fred	456-78-9012	43.0	78.0	88.0	77.0	45.0	A-
5	Rubble	Betty	234-56-7890	44.0	90.0	80.0	90.0	46.0	C-
6	Noshow	Cecil	345-67-8901	45.0	11.0	NaN	4.0	43.0	F
7	Buff	Bif	632-79-9939	46.0	20.0	30.0	40.0	50.0	B+
8	Airpump	Andrew	223-45-6789	49.0	NaN	90.0	100.0	83.0	A
9	Backus	Jim	143-12-1234	48.0	NaN	97.0	96.0	97.0	A+

Cleaning: Drop missing values

```
# drop all rows with missing values
data2.dropna()
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade
0	Alfalfa	Aloysius	123-45-6789	40.0	90.0	100.0	83.0	49.0	D-
1	Alfred	University	123-12-1234	41.0	97.0	96.0	97.0	48.0	D+
2	Gerty	Gramma	567-89-0123	41.0	80.0	60.0	40.0	44.0	C
3	Android	Electric	087-65-4321	42.0	23.0	36.0	45.0	47.0	B-
4	Bumpkin	Fred	456-78-9012	43.0	78.0	88.0	77.0	45.0	A-
5	Rubble	Betty	234-56-7890	44.0	90.0	80.0	90.0	46.0	C-
7	Buff	Bif	632-79-9939	46.0	20.0	30.0	40.0	50.0	B+
11	Dandy	Jim	087-75-4321	47.0	25.0	23.0	36.0	45.0	C+
12	Elephant	Ima	456-71-9012	45.0	29.0	78.0	88.0	77.0	B-
13	Franklin	Benny	234-56-2890	50.0	10.0	90.0	80.0	90.0	B-
15	Heffalump	Harvey	632-79-9439	30.0	16.0	20.0	30.0	40.0	C

Cleaning: Fill in missing values

```
# fill missing values in with a particular value
data2.fillna(0)
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade
0	Alfalfa	Aloysius	123-45-6789	40.0	90.0	100.0	83.0	49.0	D-
1	Alfred	University	123-12-1234	41.0	97.0	96.0	97.0	48.0	D+
2	Gerty	Gramma	567-89-0123	41.0	80.0	60.0	40.0	44.0	C
3	Android	Electric	087-65-4321	42.0	23.0	36.0	45.0	47.0	B-
4	Bumpkin	Fred	456-78-9012	43.0	78.0	88.0	77.0	45.0	A-
5	Rubble	Betty	234-56-7890	44.0	90.0	80.0	90.0	46.0	C-
6	Noshow	Cecil	345-67-8901	45.0	11.0	0.0	4.0	43.0	F
7	Buff	Bif	632-79-9939	46.0	20.0	30.0	40.0	50.0	B+
8	Airpump	Andrew	223-45-6789	49.0	0.0	90.0	100.0	83.0	A
9	Backus	Jim	143-12-1234	48.0	0.0	97.0	96.0	97.0	A+
10	Carnivore	Art	565-89-0123	44.0	0.0	80.0	60.0	40.0	D+
11	Dandy	Jim	087-75-4321	47.0	25.0	23.0	36.0	45.0	C+

Cleaning: Fill in missing values

```
# fill in missing values with average score
test2avg = data2.Test2.mean()
data2.Test2 = data2.Test2.fillna(test2avg)
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade
0	Alfalfa	Aloysius	123-45-6789	40.0	90.000000	100.0	83.0	49.0	D-
1	Alfred	University	123-12-1234	41.0	97.000000	96.0	97.0	48.0	D+
2	Gerty	Gramma	567-89-0123	41.0	80.000000	60.0	40.0	44.0	C
3	Android	Electric	087-65-4321	42.0	23.000000	36.0	45.0	47.0	B-
4	Bumpkin	Fred	456-78-9012	43.0	78.000000	88.0	77.0	45.0	A-
5	Rubble	Betty	234-56-7890	44.0	90.000000	80.0	90.0	46.0	C-
6	Noshow	Cecil	345-67-8901	45.0	11.000000	NaN	4.0	43.0	F
7	Buff	Bif	632-79-9939	46.0	20.000000	30.0	40.0	50.0	B+
8	Airpump	Andrew	223-45-6789	49.0	46.384615	90.0	100.0	83.0	A
9	Backus	Jim	143-12-1234	48.0	46.384615	97.0	96.0	97.0	A+
10	Carnivore	Art	565-89-0123	44.0	46.384615	80.0	60.0	40.0	D+
11	Dandy	Jim	087-75-4321	47.0	25.000000	23.0	36.0	45.0	C+

Aggregating: Group by

```
# drop +/- from grades
tmp = data2.Grade
tmp2 = tmp.str.replace('-','')
tmp3 = tmp2.str.replace('+','')
data2.Grade = tmp3
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade
0	Alfalfa	Aloysius	123-45-6789	40.0	90.0	100.0	83.0	49.0	D
1	Alfred	University	123-12-1234	41.0	97.0	96.0	97.0	48.0	D
2	Gerty	Gramma	567-89-0123	41.0	80.0	60.0	40.0	44.0	C
3	Android	Electric	087-65-4321	42.0	23.0	36.0	45.0	47.0	B
4	Bumpkin	Fred	456-78-9012	43.0	78.0	88.0	77.0	45.0	A
5	Rubble	Betty	234-56-7890	44.0	90.0	80.0	90.0	46.0	C
6	Noshow	Cecil	345-67-8901	45.0	11.0	NaN	4.0	43.0	F
7	Buff	Bif	632-79-9939	46.0	20.0	30.0	40.0	50.0	B
8	Airpump	Andrew	223-45-6789	49.0	NaN	90.0	100.0	83.0	A
9	Backus	Jim	143-12-1234	48.0	NaN	97.0	96.0	97.0	A
10	Carnivore	Art	565-89-0123	44.0	NaN	80.0	60.0	40.0	D
11	Dandy	Jim	087-75-4321	47.0	25.0	23.0	36.0	45.0	C

```
# find unique values
data2.Grade.unique()
array(['D', 'C', 'B', 'A', 'F'], dtype=object)
```

Aggregating: Group by

- When you iterate over the results of a groupby, each result is a tuple:
  - First element is a unique value
  - Second element is a DataFrame filtered by that value

```
# group by final grade and calculate avg Test1 score
for grade, grade_data in data2.groupby("Grade"):
    print(grade, grade_data.Test1.mean())
A 46.666666666666664
B 44.6
C 40.5
D 41.666666666666664
F 45.0
```

Aggregating: Group by

```
# group by Test1 scores and count number of students with that score
for grade, grade_data in data2.groupby("Test1"):
    print(grade, grade_data.Lastname.count())
30.0 1
40.0 2
41.0 2
42.0 1
43.0 1
44.0 2
45.0 2
46.0 1
47.0 1
48.0 1
49.0 1
50.0 1
```

Aggregating: Group by

```
# can also aggregate directly over group object
groups = data2.groupby("Grade")
groups.agg('sum')
```

	Test1	Test2	Test3	Test4	Final
Grade					
A	140.0	170.769231	275.0	273.0	225.0
B	223.0	116.000000	245.0	253.0	268.0
C	162.0	211.000000	183.0	196.0	175.0
D	125.0	233.384615	276.0	240.0	137.0
F	45.0	11.000000	NaN	4.0	43.0

```
# applying multiple aggregators
groups.Final.agg(['min', 'max'])
```

	min	max
Grade		
A	45.0	97.0
B	4.0	90.0
C	40.0	46.0
D	40.0	49.0
F	43.0	43.0

Transforming: Map

```
# map() maps values of Series according to input correspondence (defined by
dict) or function
data2['gpa'] = data2.Grade.map({'A':4.0, 'B':3.0, 'C':2.0, 'D':1.0, 'F':0.0})
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade	GPA
0	Alfalfa	Aloysius	123-45-6789	40.0	90.0	100.0	83.0	49.0	D	1.0
1	Alfred	University	123-12-1234	41.0	97.0	96.0	97.0	48.0	D	1.0
2	Gerty	Gramma	567-89-0123	41.0	80.0	60.0	40.0	44.0	C	2.0
3	Android	Electric	087-65-4321	42.0	23.0	36.0	45.0	47.0	B	3.0
4	Bumpkin	Fred	456-78-9012	43.0	78.0	88.0	77.0	45.0	A	4.0
5	Rubble	Betty	234-56-7890	44.0	90.0	80.0	90.0	46.0	C	2.0
6	Noshow	Cecil	345-67-8901	45.0	11.0	NaN	4.0	43.0	F	0.0
7	Buff	Bif	632-79-9939	46.0	20.0	30.0	40.0	50.0	B	3.0
8	Airpump	Andrew	223-45-6789	49.0	NaN	90.0	100.0	83.0	A	4.0
9	Backus	Jim	143-12-1234	48.0	NaN	97.0	96.0	97.0	A	4.0
10	Carnivore	Art	565-89-0123	44.0	NaN	80.0	60.0	40.0	D	1.0
11	Dandy	Jim	087-75-4321	47.0	25.0	23.0	36.0	45.0	C	2.0

LAMBDA functions

- Lambda functions are functions without names, for use in situations where the function will be discarded and not used again
- Example:
  ```
  lambda x: x > 0
  ```
- The term lambda makes the temporary function, x is the parameter name, and the code after `:` denotes what to do
- Often used in map( ) or apply( ) when transforming data

Transforming: Map

```
# create a rounding function, apply to each value in Final
rndGrade = lambda x: int(x / 10) * 10
data2.Final.map(rndGrade)
```

	Lastname	Firstname	SSN	Test1	Test2	Test3	Test4	Final	Grade	GPA
0	Alfalfa	Aloysius	123-45-6789	40.0	90.0	100.0	83.0	40	D	1.0
1	Alfred	University	123-12-1234	41.0	97.0	96.0	97.0	40	D	1.0
2	Gerty	Gramma	567-89-0123	41.0	80.0	60.0	40.0	40	C	2.0
3	Android	Electric	087-65-4321	42.0	23.0	36.0	45.0	40	B	3.0
4	Bumpkin	Fred	456-78-9012	43.0	78.0	88.0	77.0	40	A	4.0
5	Rubble	Betty	234-56-7890	44.0	90.0	80.0	90.0	40	C	2.0
6	Noshow	Cecil	345-67-8901	45.0	11.0	NaN	4.0	40	F	0.0
7	Buff	Bif	632-79-9939	46.0	20.0	30.0	40.0	50	B	3.0
8	Airpump	Andrew	223-45-6789	49.0	NaN	90.0	100.0	80	A	4.0
9	Backus	Jim	143-12-1234	48.0	NaN	97.0	98.0	90	A	4.0
10	Carnivore	Art	565-89-0123	44.0	NaN	80.0	60.0	40	D	1.0
11	Dandy	Jim	087-75-4321	47.0	25.0	23.0	36.0	40	C	2.0

Transforming: Apply

```
# apply() is used to apply a function to every row in given dataframe
data2.Test4.apply('sum')
966.0

# use apply() with axis=1 to send entire row to function
data2.fillna(0).apply(lambda row: row[3] + row[4] + row[5] + row[6], axis=1)
0      313.0
1      331.0
2      221.0
3      146.0
4      286.0
5      304.0
6       60.0
. . .
```

Purdue University | College of Science

Transforming: Scaling attribute values

```
# standardize Final grade by subtracting mean and dividing by stdev
avgFinal = data2.Final.mean()
stdFinal = data2.Final.std()
data2.Final.map(lambda x: (x - avgFinal)/stdFinal)
0    -0.379480
1    -0.379480
2    -0.379480
3    -0.379480
4    -0.379480
5    -0.379480
6    -0.379480
7     0.054211
8     1.355284
. . .
```