

# ENGR 298: Engineering Analysis and Decision Making – Processing Tensile Data with Pandas

Dr. Jason Forsyth  
Department of Engineering  
James Madison University

# Generating Analysis from Data

- Previously have focused on parsing raw files to identify individual data values (heart beats, RSI value, elastic modulus, yield strength..)
- Once the raw data has been processed and organized, later pipeline stages may perform additional analysis
  - Determine which participant has the best RSI; has the largest spread;
  - Determine if materials samples are outside of typical parameters
  - Automatically classify unknown material based upon properties
- Since data is often heterogenous (both numerical and textual) will use Pandas to easily parse.

Sample_Name	Material_Type	Tensile_Strength	Fracture_Strain	Elastic_Modulus	Yield_Strength
C01A1045CR_1	1045CR	787.9853962470363	0.1614	204.48360347511277	591.0496055465012
C02A1045CR_1	1045CR	808.2818259006015	0.169	187.25251265410972	615.2701167270867
C03A1045CR_1	1045CR	780.832536990291	0.1673	206.07287355644328	581.460615688504
C04A1045CR_1	1045CR	819.5370166505515	0.1584	189.37660166056475	627.5934724063153
C05A1045CR_1	1045CR	799.7138665692523	0.1678	211.22350766615887	596.0294293494273
C06A1045CR_1	1045CR	816.81265666773	0.1682	187.38946860984245	625.1167815128414
C07A1045CR_1	1045CR	785.3230418139093	0.1699	208.5144956704921	605.7125958979988
C01A2024_1	2024	473.9663518834583	0.2641	67.91973865120738	288.4807631553222
C02A2024_1	2024	476.36932476867486	0.2176	64.8677183750668	354.61976103469306
C03A2024_1	2024	466.5187365129491	0.192	72.6194161172565	354.40768936140785
C04A2024_1	2024	458.84388068219886	0.1936	69.34127686907144	352.20943726267245
C05A2024_1	2024	463.8319793916702	0.2481	69.63816298215642	354.40768936140785
C06A2024_1	2024	464.6931363115216	0.2305	109.65850256999525	350.7303375543619
C07A2024_1	2024	446.4674305430813	0.2379	68.19989930707435	344.10884909451295
C01APMMA_1	PMMA	81.36236958062278	0.05057	3.3345726004633147	47.607104924860906
C02APMMA_1	PMMA	80.46769490450241	0.0512	3.1041642360930304	52.08096308073987
C03APMMA_1	PMMA	79.20086657554037	0.06983	2.7401631572014935	45.48952546628137
C04APMMA_1	PMMA	79.0018504894893	0.04545	3.2216074851617704	51.933645498867996
C05APMMA_1	PMMA	77.01033808438562	0.05568	3.1346379779181106	46.47209282218446
C06APMMA_1	PMMA	79.33943345032885	0.05302	2.933374650143233	53.209679918783436
C07APMMA_1	PMMA	75.5981278310394	0.05304	3.179665313074314	49.09226421078233

Pandas

While pandas adopts many coding idioms from NumPy, the biggest difference is that pandas is designed for working with tabular or heterogeneous data. NumPy, by contrast, is best suited for working with homogeneous numerical array data.

# Two Data Types: Series and Data Frame

## **Series**

A Series is a one-dimensional array-like object containing a sequence of values (of similar types to NumPy types) and an associated array of data labels, called its *index*. The simplest Series is formed from only an array of data:

## **DataFrame**

A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.). The DataFrame has both a row and column index; it can be thought of as a dict of Series all sharing the same index. Under the hood, the data is stored as one or more

A **two-dimensional** labeled data structure  
with columns of potentially different types

Columns →		Country	Capital	Population
Index →	0	Belgium	Brussels	11190846
	1	India	New Delhi	1303171035
	2	Brazil	Brasilia	207847528

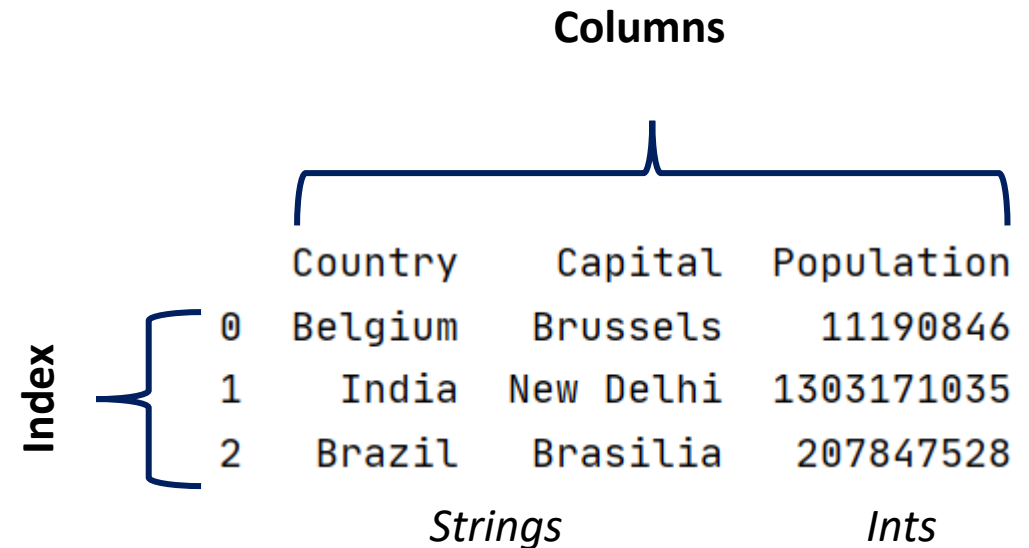
```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],  
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],  
            'Population': [11190846, 1303171035, 207847528]}  
>>> df = pd.DataFrame(data,  
                        columns=['Country', 'Capital', 'Population'])
```

# Creating and Printing a DataFrame

```
# copy in the data frame from the DataCamp guide
data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
        'Population': [11190846, 1303171035, 207847528]}

# load raw data as data frame
df = pd.DataFrame(data)

# print out the whole data frame
print(df)
```



The diagram illustrates the structure of a DataFrame. A horizontal bracket above the column headers is labeled "Columns". A vertical bracket to the left of the row indices is labeled "Index". The data is presented in a table with three columns: "Country", "Capital", and "Population". The first two columns are grouped under the label "Strings", and the third column is labeled "Ints".

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

## DataFrame

A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.). The DataFrame has both a row and column index; it can be thought of as a dict of Series all sharing the same index. Under the hood, the data is stored as one or more






# Selecting columns is pretty easy...

```
# select the 'Country' column, should generate a Series  
countries = df['Country']  countries: (0, 'Belgium') (1, 'India') (2, 'Brazil')
```

```
# print out all countries  
print(countries)
```

```
0    Belgium  
1      India  
2     Brazil
```

```
>  countries = {Series: (3,)} (0, 'Belgium') (1, 'India') (2, 'Brazil') ...View as Series  
>  data = {dict: 3} {'Country': ['Belgium', 'India', 'Brazil'], 'Capital': ['Brussels', 'New Delhi', 'Brasilia'], 'Population': [11190846, 1303171035, 207847528]}  
>  df = {DataFrame: (3, 3)} Country Capital Population [0: Belgium Brussels 11190846], [1: India New Delhi 1303171035], [2: Brazil Brasilia 207847528]
```

# Can perform more complex selecting...

```
# can perform more complex selection. Choose all countries where the capital population  
# is greater than 15 Million people. Should result in New Delhi and Brasilia  
large_capitals = df[df['Population'] > 15000000]  
  
# print out cities with large capitals  
print(large_capitals)
```

	Country	Capital	Population
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

**Syntax for 'Boolean selection' in Pandas is odd. Can replace > conditional with any conditional test. But structure of df[df[]] is the same..**

# Can perform more complex selecting...

```
# for another boolean selection, grab all data where the country is Brazil  
brazil_data = df[df['Country'] == 'Brazil']  
  
# print out brazil information  
print(brazil_data)
```

---

	Country	Capital	Population
2	Brazil	Brasilia	207847528

**Syntax for 'Boolean selection' in Pandas is odd. Can replace > conditional with any conditional test. But structure of df[df[]] is the same..**

# Converting into Numpy Arrays

```
# once an column of interest is identified, it is easy enough to convert into a numpy array  
np_array = df['Population'].to_numpy()
```

```
# should result in column vector. Print out.  
print(np_array)
```

# Comparison with Pandas and Numpy

- Both can load data from files easily but only Pandas can handle heterogeneous data.
- Pandas does have arithmetic/calculation abilities, but Numpy is superior for more advanced tasks.
- Personally, I find locating specific values/cells cumbersome in Pandas but easier to download select based upon conditions.
- Generally, load complex data with Pandas, extract what is needed, and perform calculations in Numpy.

# Example: Working with Tensile Data in Pandas

Sample_Name	Material_Type	Tensile_Strength	Fracture_Strain	Elastic_Modulus	Yield_Strength
C01A1045CR_1	1045CR	787.9853962470363	0.1614	204.48360347511277	591.0496055465012
C02A1045CR_1	1045CR	808.2818259006015	0.169	187.25251265410972	615.2701167270867
C03A1045CR_1	1045CR	780.832536990291	0.1673	206.07287355644328	581.460615688504
C04A1045CR_1	1045CR	819.5370166505515	0.1584	189.37660166056475	627.5934724063153
C05A1045CR_1	1045CR	799.7138665692523	0.1678	211.22350766615887	596.0294293494273
C06A1045CR_1	1045CR	816.81265666773	0.1682	187.38946860984245	625.1167815128414
C07A1045CR_1	1045CR	785.3230418139093	0.1699	208.5144956704921	605.7125958979988
C01A2024_1	2024	473.9663518834583	0.2641	67.91973865120738	288.4807631553222
C02A2024_1	2024	476.36932476867486	0.2176	64.8677183750668	354.61976103469306
C03A2024_1	2024	466.5187365129491	0.192	72.6194161172565	354.40768936140785
C04A2024_1	2024	458.84388068219886	0.1936	69.34127686907144	352.20943726267245
C05A2024_1	2024	463.8319793916702	0.2481	69.63816298215642	354.40768936140785
C06A2024_1	2024	464.6931363115216	0.2305	109.65850256999525	350.7303375543619
C07A2024_1	2024	446.4674305430813	0.2379	68.19989930707435	344.10884909451295
C01APMMA_1	PMMA	81.36236958062278	0.05057	3.3345726004633147	47.607104924860906
C02APMMA_1	PMMA	80.46769490450241	0.0512	3.1041642360930304	52.08096308073987
C03APMMA_1	PMMA	79.20086657554037	0.06983	2.7401631572014935	45.48952546628137
C04APMMA_1	PMMA	79.0018504894893	0.04545	3.2216074851617704	51.933645498867996
C05APMMA_1	PMMA	77.01033808438562	0.05568	3.1346379779181106	46.47209282218446
C06APMMA_1	PMMA	79.33943345032885	0.05302	2.933374650143233	53.209679918783436
C07APMMA_1	PMMA	75.5981278310394	0.05304	3.179665313074314	49.09226421078233

# Loaded as DataFrame in PyCharm

	Sample_Name	Material_Type	Tensile_Strength	Fracture_Strain	Elastic_Modulus	Yield_Strength
0	C01A1045CR_1	1045CR	787.98540	0.16140	204.48360	591.04961
1	C02A1045CR_1	1045CR	808.28183	0.16900	187.25251	615.27012
2	C03A1045CR_1	1045CR	780.83254	0.16730	206.07287	581.46062
3	C04A1045CR_1	1045CR	819.53702	0.15840	189.37660	627.59347
4	C05A1045CR_1	1045CR	799.71387	0.16780	211.22351	596.02943
5	C06A1045CR_1	1045CR	816.81266	0.16820	187.38947	625.11678
6	C07A1045CR_1	1045CR	785.32304	0.16990	208.51450	605.71260
7	C01A2024_1	2024	473.96635	0.26410	67.91974	288.48076
8	C02A2024_1	2024	476.36932	0.21760	64.86772	354.61976
9	C03A2024_1	2024	466.51874	0.19200	72.61942	354.40769
10	C04A2024_1	2024	458.84388	0.19360	69.34128	352.20944
11	C05A2024_1	2024	463.83198	0.24810	69.63816	354.40769
12	C06A2024_1	2024	464.69314	0.23050	109.65850	350.73034
13	C07A2024_1	2024	446.46743	0.23790	68.19990	344.10885
14	C01APMMA_1	PMMA	81.36237	0.05057	3.33457	47.60710
15	C02APMMA_1	PMMA	80.46769	0.05120	3.10416	52.08096
16	C03APMMA_1	PMMA	79.20087	0.06983	2.74016	45.48953
17	C04APMMA_1	PMMA	79.00185	0.04545	3.22161	51.93365
18	C05APMMA_1	PMMA	77.01034	0.05568	3.13464	46.47209



Scenario: How would you go about calculating the  $\mu \pm \sigma$  for each of each property for all material types...

	Sample_Name	Material_Type	Tensile_Strength	Fracture_Strain	Elastic_Modulus	Yield_Strength
0	C01A1045CR_1	1045CR	787.98540	0.16140	204.48360	591.04961
1	C02A1045CR_1	1045CR	808.28183	0.16900	187.25251	615.27012
2	C03A1045CR_1	1045CR	780.83254	0.16730	206.07287	581.46062
3	C04A1045CR_1	1045CR	819.53702	0.15840	189.37660	627.59347
4	C05A1045CR_1	1045CR	799.71387	0.16780	211.22351	596.02943
5	C06A1045CR_1	1045CR	816.81266	0.16820	187.38947	625.11678
6	C07A1045CR_1	1045CR	785.32304	0.16990	208.51450	605.71260
7	C01A2024_1	2024	473.96635	0.26410	67.91974	288.48076
8	C02A2024_1	2024	476.36932	0.21760	64.86772	354.61976
9	C03A2024_1	2024	466.51874	0.19200	72.61942	354.40769
10	C04A2024_1	2024	458.84388	0.19360	69.34128	352.20944
11	C05A2024_1	2024	463.83198	0.24810	69.63816	354.40769
12	C06A2024_1	2024	464.69314	0.23050	109.65850	350.73034
13	C07A2024_1	2024	446.46743	0.23790	68.19990	344.10885
14	C01APMMA_1	PMMA	81.36237	0.05057	3.33457	47.60710
15	C02APMMA_1	PMMA	80.46769	0.05120	3.10416	52.08096
16	C03APMMA_1	PMMA	79.20087	0.06983	2.74016	45.48953
17	C04APMMA_1	PMMA	79.00185	0.04545	3.22161	51.93365
18	C05APMMA_1	PMMA	77.01034	0.05568	3.13464	46.47209

# Using Pandas to select based upon Material

*# get all results from one type*

```
cold_rolled = df[df['Material_Type'] == '1045CR']
```

```
stainless_steel = df[df['Material_Type'] == '2024']
```

```
plastic = df[df['Material_Type'] == 'PMMA']
```

*# print out each material type*

```
print(cold_rolled)
```

```
print(stainless_steel)
```

```
print(plastic)
```

	Sample_Name	Material_Type	...	Elastic_Modulus	Yield_Strength
0	C01A1045CR_1	1045CR	...	204.483603	591.049606
1	C02A1045CR_1	1045CR	...	187.252513	615.270117
2	C03A1045CR_1	1045CR	...	206.072874	581.460616
3	C04A1045CR_1	1045CR	...	189.376602	627.593472
4	C05A1045CR_1	1045CR	...	211.223508	596.029429
5	C06A1045CR_1	1045CR	...	187.389469	625.116782
6	C07A1045CR_1	1045CR	...	208.514496	605.712596

[7 rows x 6 columns]

	Sample_Name	Material_Type	...	Elastic_Modulus	Yield_Strength
7	C01A2024_1	2024	...	67.919739	288.480763
8	C02A2024_1	2024	...	64.867718	354.619761
9	C03A2024_1	2024	...	72.619416	354.407689
10	C04A2024_1	2024	...	69.341277	352.209437
11	C05A2024_1	2024	...	69.638163	354.407689
12	C06A2024_1	2024	...	109.658503	350.730338
13	C07A2024_1	2024	...	68.199899	344.108849

[7 rows x 6 columns]

	Sample_Name	Material_Type	...	Elastic_Modulus	Yield_Strength
14	C01APMMA_1	PMMA	...	3.334573	47.607105
15	C02APMMA_1	PMMA	...	3.104164	52.080963
16	C03APMMA_1	PMMA	...	2.740163	45.489525
17	C04APMMA_1	PMMA	...	3.221607	51.933645
18	C05APMMA_1	PMMA	...	3.134638	46.472093
19	C06APMMA_1	PMMA	...	2.933375	53.209680
20	C07APMMA_1	PMMA	...	3.179665	49.092264

# Performing Analysis with Pandas

- Since analysis is simple (just  $\mu \pm \sigma$ ) can use built-in Pandas mean() and std() functions for each DataFrame.
- If analysis is more complex then would port each array in Numpy
- Following example will be a “long”/manual version, will show later example that adjusts for arbitrary number of materials

As data is sorted by material, need to isolate each 'property' column and perform analysis

```
# determine averages/deviations on elastic modulus of each material  
cr_modulus_avg = cold_rolled['Elastic_Modulus'].mean()  
cr_modulus_std = cold_rolled['Elastic_Modulus'].std()  
  
ss_modulus_avg = stainless_steel['Elastic_Modulus'].mean()  
ss_modulus_std = stainless_steel['Elastic_Modulus'].std()  
  
pmma_modulus_avg = plastic['Elastic_Modulus'].mean()  
pmma_modulus_std = plastic['Elastic_Modulus'].std()
```

*For each material that we know (cold rolled, stainless steel, and plastic) determine the  $\mu$  and  $\sigma$  via Pandas. Store the result in a variable.*

```
# determine averages/deviations on yield strength of each material
cr_yieldstrength_avg = cold_rolled['Yield_Strength'].mean()
cr_yieldstrength_std = cold_rolled['Yield_Strength'].std()

ss_yieldstrength_avg = stainless_steel['Yield_Strength'].mean()
ss_yieldstrength_std = stainless_steel['Yield_Strength'].std()

pmma_yieldstrength_avg = plastic['Yield_Strength'].mean()
pmma_yieldstrength_std = plastic['Yield_Strength'].std()
```

*Perform analysis again but with Yield Strength.*

# Print out all results in a nice table

```
# print out results
print("Material\tElastic Modulus\tYield Strength")

print("1045CR\t", round(cr_modulus_avg, 2), "+/-", round(cr_modulus_std, 2),
      "\t", round(cr_yieldstrength_avg, 2), "+/-", round(cr_yieldstrength_std, 2))

print("2024\t", round(ss_modulus_avg, 2), "+/-", round(ss_modulus_std, 2),
      "\t", round(ss_yieldstrength_avg, 2), "+/-", round(ss_yieldstrength_std, 2))

print("PMMA\t", round(pmma_modulus_avg, 2), "+/-", round(pmma_modulus_std, 2),
      "\t", round(pmma_yieldstrength_avg, 2), "+/-", round(pmma_yieldstrength_std, 2))
```

Material	Elastic Modulus	Yield Strength
1045CR	199.19 +/- 10.69	606.03 +/- 17.53
2024	74.61 +/- 15.63	342.71 +/- 24.2
PMMA	3.09 +/- 0.2	49.41 +/- 3.04

Previous approach works well but would not expand well if many more material types were added. Would need to 're-type' for each new material.

Solution: use the `unique()` function in Pandas to generate a list of all unique values in the Materials column.

*Generate a list of  
all materials*

```
# get all materials as numpy array
materials = df['Material_Type'].unique()  materials: ['1045CR' '2024' 'PMMA']

# convert to a list for easy iteration
materials = materials.tolist()
```

*Iterate through  
the list and for  
each material  
calculate  
statistics on Yield  
and Modulus*

```
# being printing table while we iterate
print("Material\tElastic Modulus\tYield Strength")

for mat in materials:
    material_statistics = df[df['Material_Type'] == mat]
    yield_strength = material_statistics['Yield_Strength']
    elastic_modulus = material_statistics['Elastic_Modulus']

    yield_avg = yield_strength.mean()
    yield_std = yield_strength.std()

    modulus_avg = elastic_modulus.mean()
    modulus_std = elastic_modulus.std()

    print(mat, "\t", round(modulus_avg, 2), "+/-", round(modulus_std, 2),
          "\t", round(yield_avg, 2), "+/-", round(yield_std, 2))

print("Done!")
```

*Mat will iterate through  
1045CR, 2024, and then  
PMMA*

*Approach will result in same table but will automatically expand if new materials are added. Would not expand if new material properties (toughness, crushing...etc.) were added.*



# Coming up this week..

- Will cover writing data to files on Wednesday then t-tests?
- Next week: curve fitting and statistical distributions
- Will be at VT on Friday for research; will generate Gradescope for this assignment later in the week.
- Gradescope for tensile data assigned

