

DATA SOCIETY®

Day 4 - Data wrangling with Python

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Load data into Python and inspect	
Summarize data using pandas	
Reshape data using pandas	

Recap

- In the last module you learned how to:
 - Work with Pandas series and dataframes
 - Read in a csv
- We will now load the same dataset from our data folder, clean it and apply functions to it



Import Pandas and os

- Let's import the pandas and os library
- Note: it is not required that you also import numpy in order to use pandas

```
import pandas as pd  
import numpy as np  
import os
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into variables
- Let the `main_dir` be the variable corresponding to your skill-soft folder

```
# Set `main_dir` to the location of your `skill-soft` folder (for Linux).  
main_dir = "/home/[username]/Desktop/skill-soft"
```

```
# Set `main_dir` to the location of your `skill-soft` folder (for Mac).  
main_dir = "/Users/[username]/Desktop/skill-soft"
```

```
# Set `main_dir` to the location of your `skill-soft` folder (for Windows).  
main_dir = "C:\\\\Users\\\\[username]\\\\Desktop\\\\skill-soft"
```

```
# Make `data_dir` from the `main_dir` and  
# remainder of the path to data directory.  
data_dir = main_dir + "/data"
```

Setting working directory

- Set working directory to data_dir

```
# Set working directory.  
os.chdir(data_dir)
```

```
# Check working directory.  
print(os.getcwd())
```

```
/home/[user-name]/Desktop/skill-soft/data
```

Read data from csv file

- We are now going to use the function `read_csv` to read in our `household_poverty` dataset
- Let's load the entire dataset

```
household_poverty = pd.read_csv('household_poverty.csv')
print(household_poverty.head())
```

```
   male      hh_ID  rooms  ...  water_inside  years_of_schooling  Target
0    1  21eb7fcc1      3  ...          1                10        4
1    1  0e5d7a658      4  ...          1                12        4
2    0  2c7317ea8      8  ...          1                11        4
3    1  2b58d945f      5  ...          1                 9        4
4    0  2b58d945f      5  ...          1                11        4
```

[5 rows x 14 columns]

Inspect data

- What have we just created? **Let's inspect**

```
print(type(household_poverty)) #<- a pandas dataframe!
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
print(len(household_poverty)) #<- returns the number of rows
```

```
1000
```

```
# You can also save the shape of the dataframe into 2 variables  
# (since the returned is a tuple with 2 values).
```

```
nrows, ncols = household_poverty.shape  
print(nrows)
```

```
1000
```

```
print(ncols)
```

```
14
```

Previewing data - using head method

- We've already used `.head()` command that shows the first few rows
- Let's inspect this data

```
print(household_poverty.head()) #<- pulls the first 5 rows (the default is 5)
```

```
   male      hh_ID  rooms  ...  water_inside  years_of_schooling  Target
0    1  21eb7fccc1      3  ...          1                  10        4
1    1  0e5d7a658      4  ...          1                  12        4
2    0  2c7317ea8      8  ...          1                  11        4
3    1  2b58d945f      5  ...          1                  9        4
4    0  2b58d945f      5  ...          1                 11        4
[5 rows x 14 columns]
```

- This is a great way to understand the data that we have without having to call the entire dataset

Previewing data - using head method

- We can specify the number of rows we want to see:

```
print(household_poverty.head(3)) #<- pulls the first 3 rows
```

	male	hh_ID	rooms	...	water_inside	years_of_schooling	Target
0	1	21eb7fcc1	3	...	1	10	4
1	1	0e5d7a658	4	...	1	12	4
2	0	2c7317ea8	8	...	1	11	4

[3 rows x 14 columns]

Previewing data - using sample method

- We can view some random rows in the dataframe by using the `.sample()` method

```
print(household_poverty.sample(n = 3))      #<- 3 random rows
```

```
   male    hh_ID  rooms  ...  water_inside  years_of_schooling  Target
47     0  6ddd55b3e     5  ...          1                  16        4
789    1  f85fc9f64     6  ...          1                  6        4
581    1  fbe69128f     5  ...          1                 15        4
[3 rows x 14 columns]
```

Previewing data - using sample method

- We can specify to see a percentage rather than an actual exact number

```
print(household_poverty.sample(frac = .02)) #<- a random 2% of the rows
```

	male	hh_ID	rooms	...	water_inside	years_of_schooling	Target
391	0	d3f101545	7	...	1	0	4
207	1	1ea163834	8	...	1	15	4
556	1	235cbd214	3	...	1	9	4
364	0	5f4043e63	5	...	1	6	4
186	1	322cefd2f	4	...	1	0	2
90	1	caa343c15	4	...	1	8	4
765	1	04b02301b	11	...	1	14	4
619	0	1239df1cc	3	...	1	9	4
840	0	5a306db94	5	...	1	11	4
957	0	0e7f79b84	9	...	1	14	4
278	0	0473832dd	5	...	1	3	4
247	0	fabb02be9	4	...	1	0	4
669	1	e8725f7cb	4	...	1	0	4
326	0	c60226aa6	6	...	1	5	4
666	0	ff31b984b	7	...	1	6	4
971	0	a1313caf6	5	...	1	2	3
738	0	0fb516a6a	5	...	1	11	4
393	1	d3f101545	7	...	1	17	4
11	0	bb2094100	3	...	1	12	4
706	1	91fc4c109	6	...	1	15	4

[20 rows x 14 columns]

Reviewing household_poverty data

- We are now going to get to know our data better, using the following pandas techniques:
 - .columns
 - .dtypes
 - .info()
 - .describe()

```
print(household_poverty.columns)
```

```
Index(['male', 'hh_ID', 'rooms', 'males_tot',  
       'age', 'ppl_total', 'num_child',  
       'bedrooms', 'dependency_rate',  
       'computer', 'disabled_ppl',  
       'water_inside', 'years_of_schooling',  
       'Target'],  
      dtype='object')
```

```
print(household_poverty.dtypes)
```

male	int64
hh_ID	object
rooms	int64
males_tot	int64
age	int64
ppl_total	int64
num_child	int64
bedrooms	int64
dependency_rate	int64
computer	int64
disabled_ppl	int64
water_inside	int64
years_of_schooling	int64
Target	int64
dtype: object	

Reviewing household_poverty data - info

```
print(household_poverty.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
male                1000 non-null int64
hh_ID               1000 non-null object
rooms               1000 non-null int64
males_tot           1000 non-null int64
age                 1000 non-null int64
ppl_total           1000 non-null int64
num_child            1000 non-null int64
bedrooms            1000 non-null int64
dependency_rate     1000 non-null int64
computer             1000 non-null int64
disabled_ppl         1000 non-null int64
water_inside          1000 non-null int64
years_of_schooling   1000 non-null int64
Target              1000 non-null int64
dtypes: int64(13), object(1)
memory usage: 109.5+ KB
None
```

Reviewing household_poverty data - describe

```
print(household_poverty.describe())
```

	male	rooms	...	years_of_schooling	Target
count	1000.000000	1000.000000	...	1000.000000	1000.000000
mean	0.477000	5.244000	...	8.055000	3.586000
std	0.499721	1.692728	...	4.898568	0.833846
min	0.000000	1.000000	...	0.000000	1.000000
25%	0.000000	4.000000	...	5.000000	4.000000
50%	0.000000	5.000000	...	8.000000	4.000000
75%	1.000000	6.000000	...	11.000000	4.000000
max	1.000000	11.000000	...	21.000000	4.000000

[8 rows x 13 columns]

Reviewing household_poverty data - index

- What is the index of this dataframe?

```
print(household_poverty.index)
```

```
RangeIndex(start=0, stop=1000, step=1)
```

- Remember, we can set the index as one of our columns
- What would make most sense with this dataset?

```
household_poverty = household_poverty.set_index('hh_ID')
print(household_poverty.index)
```

```
Index(['21eb7fcc1', '0e5d7a658', '2c7317ea8', '2b58d945f', '2b58d945f',
       '2b58d945f', '2b58d945f', 'd6dae86b7', 'd6dae86b7', 'd6dae86b7',
       ...
       'f54bc4b11', 'f54bc4b11', 'f54bc4b11', 'f54bc4b11', '34bab1f1f',
       '34bab1f1f', '34bab1f1f', '34bab1f1f', '34bab1f1f', '52cf8fe9d'],
      dtype='object', name='hh_ID', length=1000)
```

- Now we can look up rows by the actual household IDs

Looking up by household ID

- Let's refresh looking up by index
- We can use `.loc` to look up by specific household ID

```
# Look up a specific row by index.  
print(household_poverty.loc['21eb7fcc1'])
```

```
male                1  
rooms               3  
males_tot           1  
age                 43  
ppl_total            1  
num_child             0  
bedrooms              1  
dependency_rate      30  
computer                0  
disabled_ppl           0  
water_inside            1  
years_of_schooling     10  
Target                  4  
Name: 21eb7fcc1, dtype: int64
```

- **When would something like this be useful in your data?**

Looking up by household ID

- We can use `.iloc` to look up by row number of the index

```
# Look up a specific row by index.  
print(household_poverty.iloc[1])
```

```
male                1  
rooms               4  
males_tot           1  
age                 67  
ppl_total            1  
num_child             0  
bedrooms              1  
dependency_rate      29  
computer              0  
disabled_ppl          0  
water_inside           1  
years_of_schooling     12  
Target                 4  
Name: 0e5d7a658, dtype: int64
```

Reset index

- And finally, we can reset our index back to the original index

```
household_poverty = household_poverty.reset_index()
```

- In the next part, you will load a dataset into your notebook and summarize data using pandas

Knowledge check 1



Exercise 1



Module completion checklist

Objective	Complete
Load data into Python and inspect	✓
Summarize data using pandas	
Reshape data using pandas	

Methods to summarize and group data in pandas

- What if we want more detailed summary metrics? Use `groupby()`!
- `groupby()` describes a process involving the following steps:
 - **splitting** the data into groups based on a specific criteria
 - **applying** a function to each group independently
- We'll be starting with the most straightforward part of `groupby()`, the split step

Splitting using groupby()

- A string passed to `groupby()` may refer to either a column or an index level
- We can either group by column or by index
- This fits into the **splitting** step, as we are splitting the data to be grouped by number of rooms
- We will group by the column `rooms` for now

```
grouped = household_poverty.groupby('rooms')
print(grouped.first())
```

rooms	hh_ID	male	males_tot	...	water_inside	years_of_schooling	Target
1	3e16fab89	1	2	...	0	6	4
2	d6dae86b7	0	1	...	1	0	4
3	21eb7fcc1	1	1	...	1	10	4
4	0e5d7a658	1	1	...	1	12	4
5	2b58d945f	1	2	...	1	9	4
6	65d20b573	0	2	...	1	14	4
7	bcc196e5a	0	1	...	1	13	4
8	2c7317ea8	0	0	...	1	11	4
9	6f1edad4e	1	3	...	1	13	4
10	bdd842cf8	1	2	...	1	11	4
11	b64f4194f	1	1	...	1	9	4

[11 rows x 13 columns]

Summarizing using groupby()

- All the summary functions can be applied to a group
- Here are the summary functions:

Function	Description
count	Number of non-null observations
sum	Number of non-null observations
max	Maximum of values
min	Minimum of values
mean	Mean of values
median	Arithmetic median of values
var	Variance of each object
std	Standard deviation of each object

Groupby() and summary functions

- We can now move to the second step of summarizing data, **applying** a function to the group
- Let's see the distribution of households by room

```
# We are counting the number of hh_IDs by number
# of rooms, and creating a dataframe.
hh_ID = grouped.count()[['hh_ID']]
print(hh_ID)
```

```
hh_ID
rooms
1      8
2     19
3     90
4    227
5    291
6    156
7    112
8      59
9      14
10     17
11      7
```

```
# This syntax would do the same, but create a
# series.
print(grouped.count()['hh_ID'])
```

```
rooms
1      8
2     19
3     90
4    227
5    291
6    156
7    112
8      59
9      14
10     17
11      7
Name: hh_ID, dtype: int64
```

A little more about summarizing - sorting

- **Sorting**: ordering row(s) by value of column, either low to high (default) or high to low (ascending = False)

```
print(hh_ID.sort_values(by = ['hh_ID'], ascending = [False]))
```

rooms	hh_ID
5	291
4	227
6	156
7	112
3	90
8	59
2	19
10	17
9	14
1	8
11	7

A little more about summarizing - filtering

- **Filtering:** using filter(s) to subset only certain aspects of your dataset

```
print(hh_ID.query('rooms < 5'))
```

rooms	hh_ID
1	8
2	19
3	90
4	227

A little more about summarizing - new columns

- **Create new columns** by creating a series and adding it to a current dataframe

```
over100_hh = hh_ID['hh_ID'] > 100
# Add the new column.
hh_ID['over100_hh'] = over100_hh
print(hh_ID.head())
```

rooms	hh_ID	over100_hh
1	8	False
2	19	False
3	90	False
4	227	True
5	291	True

Knowledge check 2



Exercise 2



Module completion checklist

Objective	Complete
Load data into Python and inspect	✓
Summarize data using pandas	✓
Reshape data using pandas	

Data wrangling and exploration

- Remember, a data scientist must be able to:
 - **Wrangle** the data (gather, clean, and sample data to get a suitable data set)
 - **Manage** the data for easy access by the organization
 - **Explore** the data to generate a hypothesis
- **The techniques we will learn in this course will help us achieve these goals!**
- **We will work with the entire Costa Rican dataset and see what we can discover. We will be:**
 - Cleaning the dataset
 - Wrangling the data for the purpose of visualizing the data and identifying patterns

Load the dataset

- Let's load the entire dataset
- For reshaping, we will be taking a specific subset
- We are now going to use the function `read_csv` to read in our `costa_rican_poverty` dataset

```
household_poverty = pd.read_csv("costa_rica_poverty.csv")
print(household_poverty.head())
```

```
household_id      ind_id rooms ... age Target monthly_rent
0   21eb7fcc1  ID_279628684    3 ... 43     4  190000.0
1   0e5d7a658  ID_f29eb3ddd    4 ... 67     4  135000.0
2   2c7317ea8  ID_68de51c94    8 ... 92     4        NaN
3   2b58d945f  ID_d671db89c    5 ... 17     4  180000.0
4   2b58d945f  ID_d56d6f5f5    5 ... 37     4  180000.0

[5 rows x 84 columns]
```

- The entire dataset consists of 9557 observations and 84 variables

Subsetting data

- We will explore a subset of this dataset, which includes the following variables:
 - **ppl_total**
 - **dependency_rate**
 - **num_adults**
 - **monthly rent**
 - **rooms**
 - **age**
 - *Target*
- We are choosing these variables because they illustrate the concepts best
- However, you should be able to work with (and visualize) all of your data

Subsetting data

- Let's subset our data so that we have the variables we need
- We are keeping ppl_total, dependency_rate, num_adults, rooms, age, monthly_rent, and Target
- Let's name this subset costa_viz

```
costa_viz = household_poverty[['ppl_total', 'dependency_rate',
                               'num_adults', 'rooms', 'age', 'monthly_rent',
                               'Target']]
print(costa_viz.head())
```

	ppl_total	dependency_rate	num_adults	rooms	age	monthly_rent	Target
0	1	37	1	3	43	190000.0	4
1	1	36	1	4	67	135000.0	4
2	1	36	1	8	92	NaN	4
3	4	38	2	5	17	180000.0	4
4	4	38	2	5	37	180000.0	4

Data prep: clean NAs

- Depending on **subject matter**, missing values might mean something
- Let's define the choices on **how we can handle NAs in our data:**
 - drop columns that contain any NAs
 - drop columns with a certain % of NAs
 - impute missing values
 - convert column with missing values to categorical
- Let's look at the count of NAs by column first:

```
print(costas_viz.isnull().sum())
```

```
ppl_total          0
dependency_rate    0
num_adults         0
rooms              0
age                 0
monthly_rent       6860
Target              0
dtype: int64
```

Data cleaning: NAs

- monthly_rent has many NA values!
- We could just drop this column, as the number is over 50%
- However, in this instance, we'll keep it, and **impute missing values** using the mean of the column
- There isn't a mathematical method for a precise percentage of NAs that we are OK with
- **That's why your subject matter expertise is so important!**

```
# Set the dataframe equal to the imputed dataset.  
costa_viz = costa_viz.fillna(costा_viz.mean())  
# Check how many values are null in monthly_rent.  
print(costa_viz.isnull().sum())
```

```
ppl_total      0  
dependency_rate 0  
num_adults     0  
rooms          0  
age             0  
monthly_rent    0  
Target          0  
dtype: int64
```

Converting the target variable

- Let's convert poverty to a variable with two levels, which will help to balance it out
- The four original levels would also increase the complexity of the visualizations and the code
- For this reason, we will convert levels 1, 2 and 3 to `vulnerable` and 4 to `non_vulnerable`
- The levels translate to 1, 2 and 3 as being **vulnerable** households
- Level 4 is **non-vulnerable**

```
costa_viz['Target'] = np.where(costa_viz['Target'] <= 3, 'vulnerable', 'non_vulnerable')
```

```
print(costa_viz['Target'].head())
```

```
0    non_vulnerable
1    non_vulnerable
2    non_vulnerable
3    non_vulnerable
4    non_vulnerable
Name: Target, dtype: object
```

Data prep: target

- The next step of our data cleanup is to ensure the target variable is binary and has a label
- Let's look at the dtype of Target

```
print(costa_viz.Target.dtypes)
```

```
object
```

- We want to convert this to bool so that it is a binary class

```
costa_viz["Target"] = np.where(costa_viz["Target"] == "non_vulnerable", True, False)  
# Check class again.  
print(costa_viz.Target.dtypes)
```

```
bool
```

Data reshaping: wide vs long

- When we talk about data *reshaping*, what we usually mean is converting between what is called either **wide** or **long** data format
 - Wide** data is much more visually digestible, which is why you're likely to come across it if you are using data from some type of report
 - Long** data is much easier to work with in Pandas, and generally speaking in most data analysis and plotting tools

Data reshaping: wide vs long (cont'd)

- **Wide** data often appears when the values are some type of aggregate (we will use mean of groups)
- Let's make a dataframe with two rows and six columns that looks like this, it represents a typical **wide** dataframe

Target	ppl_total	dependency_rate	num_adults	rooms	age
False	4.358607	26.011233	2.388093	4.533839	31.314238
True	3.796531	25.425284	2.713809	5.205971	36.078886

Prepare data: group and summarize

- Now that we know how to group and summarize data, let's create a summary dataset that would include the following:
- Grouped data by Target variable
- Mean value computed on the grouped data that includes the following variables:
 - ppl_total
 - dependency_rate
 - num_adults
 - rooms
 - age

Prepare data: group and summarize (cont'd)

```
# Group data by `Target` variable.  
grouped = costa_viz.groupby('Target')
```

```
# Compute mean on the listed variables using the grouped data.  
costa_grouped_mean = grouped.mean() [['ppl_total', 'dependency_rate', 'num_adults', 'rooms', 'age']]  
print(costa_grouped_mean)
```

Target	ppl_total	dependency_rate	num_adults	rooms	age
False	4.358607	26.011233	2.388093	4.533839	31.314238
True	3.796531	25.425284	2.713809	5.205971	36.078886

Prepare data: group and summarize (cont'd)

```
# Reset index of the dataset.  
costa_grouped_mean = costa_grouped_mean.reset_index()  
print(costa_grouped_mean)
```

	Target	ppl_total	dependency_rate	num_adults	rooms	age
0	False	4.358607	26.011233	2.388093	4.533839	31.314238
1	True	3.796531	25.425284	2.713809	5.205971	36.078886

- The reason we call this dataframe **wide** is because each variable has its own column (i.e. `ppl_total`, `age`, etc.)
- It makes the table easier to present, but is inconvenient to run analyses on or visualize

Why long?

- Now let's convert this **wide** data to the **long** format
- The metric variable, which was previously presented in 5 columns (i.e. ppl_total, age, etc), should be put into a single column
- The mean variable was the values in the columns corresponding to those variables
- That's the format we expect to get when we convert our **wide** dataframe to **long**
- This format is very convenient to work with when we run analysis and plot data

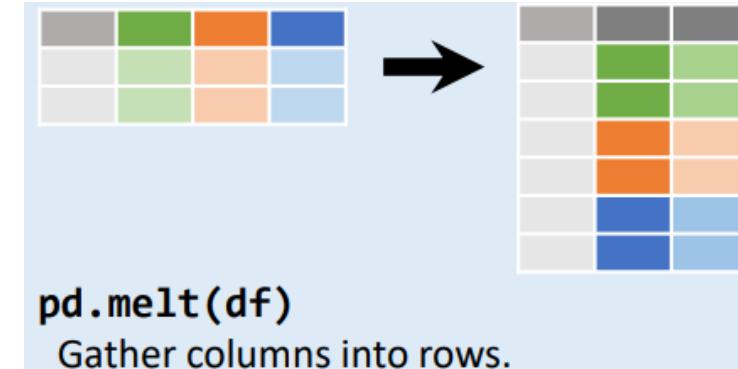
mrc_class	metric	mean
False	ppl_total	4.358607
True	ppl_total	3.796531
False	dependency_rate	26.011233
True	dependency_rate	25.425284
False	num_adults	2.388093
True	num_adults	2.713809
False	rooms	4.533839
True	rooms	5.205971
False	age	31.314238
True	age	36.078886

Wide to long format: melt

To convert from **wide** to **long** format, we use the Pandas `melt` function with the following arguments:

1. Wide dataframe
2. Variable(s) that will be preserved as the ids of the data (i.e. like Target with values True and False in our case)
3. Name of the variable that will now contain the column names from the wide data we want to melt together
4. Name of the column that will contain respective values corresponding to the melted columns

```
pd.melt(df,  
        id_vars = ['id_col'],      #<- 1  
        var_name = 'some_var',    #<- 2  
        value_name = 'some_val') #<- 3
```



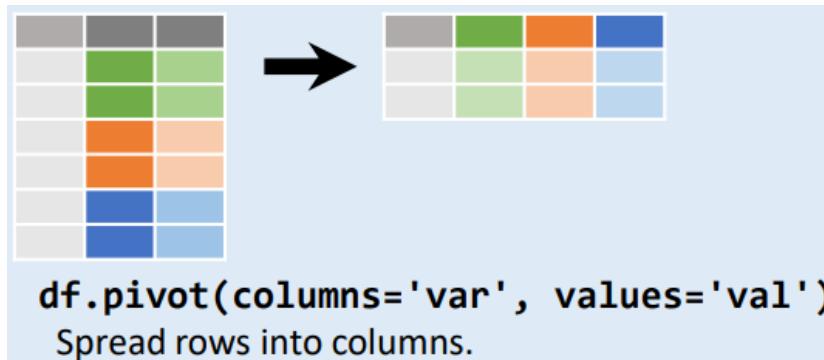
Wide to long format: melt (cont'd)

```
# Melt the wide data into long.
costa_grouped_mean_long = pd.melt(costa_grouped_mean,
                                   id_vars = ['Target'],
                                   var_name = 'metric',
                                   value_name = 'mean') #<- wide dataset
#<- identifying variable
#<- contains col names of wide data
#<- contains values from above columns
print(costa_grouped_mean_long)
```

```
   Target      metric      mean
0  False  ppl_total  4.358607
1  True  ppl_total  3.796531
2  False  dependency_rate  26.011233
3  True  dependency_rate  25.425284
4  False  num_adults  2.388093
5  True  num_adults  2.713809
6  False        rooms  4.533839
7  True        rooms  5.205971
8  False        age  31.314238
9  True        age  36.078886
```

Long to wide format: pivot

```
df.pivot(index = ['id_col'],      #<- 1  
         columns = 'some_var',    #<- 2  
         values = 'some_val')    #<- 3
```



We can convert the **long** data back to **wide** format with the `.pivot()` method

1. The `index` argument refers to what values *will become* the ids in the new dataframe
2. The `columns` argument refers to the values of which column will be converted to column names
3. Lastly, we supply the `values` argument, which is the field to use to fill in the values of the wide data

Long to wide format: pivot (cont'd)

```
# Melt the long data into wide.
costa_grouped_mean_wide = costa_grouped_mean_long.pivot(
    index = 'Target',      #<- identifying variable
    columns = 'metric',    #<- col names of wide data
    values = 'mean')       #<- values from above columns
print(costa_grouped_mean_wide)
```

metric	age	dependency_rate	num_adults	ppl_total	rooms
Target					
False	31.314238	26.011233	2.388093	4.358607	4.533839
True	36.078886	25.425284	2.713809	3.796531	5.205971

Knowledge check 3



Exercise 3



Module completion checklist

Objective	Complete
Load data into Python and inspect	✓
Summarize data using pandas	✓
Reshape data using pandas	✓

Summary

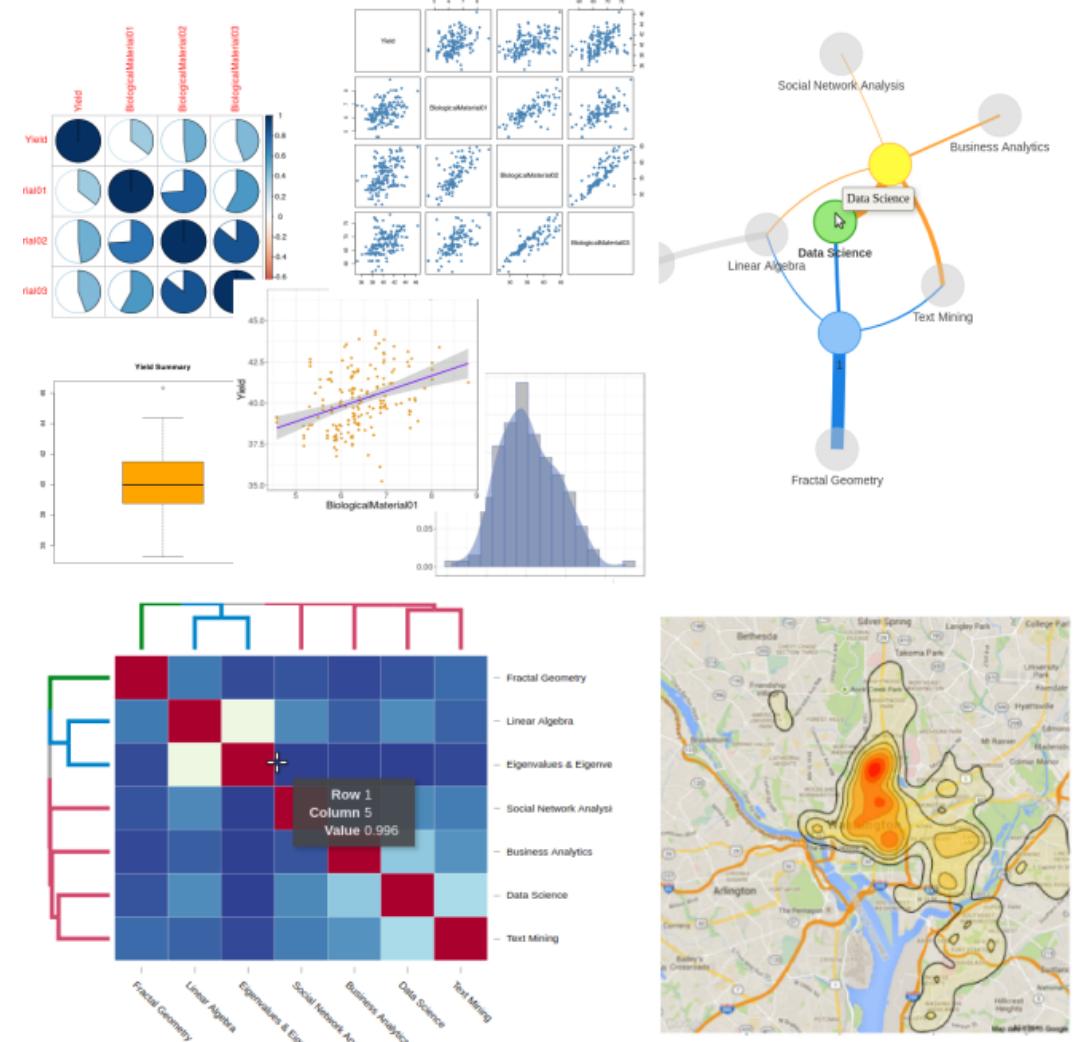
- We have covered the following throughout this course:

Objective	Complete
Data science life cycle and use cases	✓
Working with Numpy arrays	✓
Applying operations on Pandas series and dataframe	✓
Working with a dataset and reshaping variables	✓

What next?

The next steps after cleaning the data is to try out:

- **Exploratory data analysis** using visualizations
 - EDA is commonly the first step taken after we encounter a dataset
 - It's important to perform EDA as it gives us an idea on how to deal with the data and gain insights from it

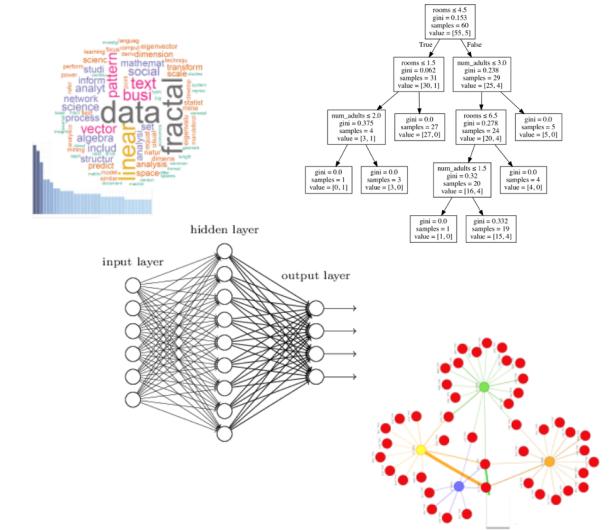


What next?

- **Data modeling**

- Supervised machine learning is the task of predicting or classifying an output based on predictors, or input variables
- Below are some examples of how you would apply these algorithms in business scenarios

Question to answer	Real world example
What is the value based on predictors?	Predicting the number of bikes rented based on the weather
What category is this in?	Anticipating if your customer is pregnant, remodeling, just got married, etc.
What is the probability that something is in a given category?	Determining the probability that a piece of equipment will fail, or that someone will buy your product



This completes our module
Congratulations!

