DATA SOCIETY®

Visualization in Python - Day 1

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


Welcome

Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

Welcome: Icebreaker!

In the chat box, answer one of the following questions:

1. Post your short bio (name, position anything else you'd like to share)

2. What do you see out the nearest window?

3. How much programming experience do you have and what are you excited to learn?

Activity

- Please count how many times the number 7 appears in the grid below. Make a note of how long it takes you

5	2	8	3	6	1	9	3	6	2	5	3	7	4	3	8	3
8	5	8	9	6	2	1	4	4	3	9	3	6	5	2	4	9
1	0	2	7	5	2	8	3	6	1	6	2	9	3	8	3	8
5	8	4	7	2	0	3	7	3	5	4	7	1	8	2	0	1
2	5	3	6	4	3	9	1	0	8	9	5	7	3	4	5	3
2	7	5	2	8	3	6	1	6	2	9	3	8	3	8	5	8
4	7	2	0	3	7	3	5	4	7	1	8	2	0	1	9	6
2	1	4	4	3	9	3	6	5	2	4	9	1	0	2	7	5
2	8	3	6	1	6	2	9	3	8	3	8	5	8	4	7	2
0	3	7	3	5	4	7	1	8	2	0	1	2	5	3	6	4
3	9	1	8	9	5	0	7	3	4	5	3	2	7	5	2	8
3	6	1	6	2	4	6	2	7	5	9	1	5	2	6	3	6

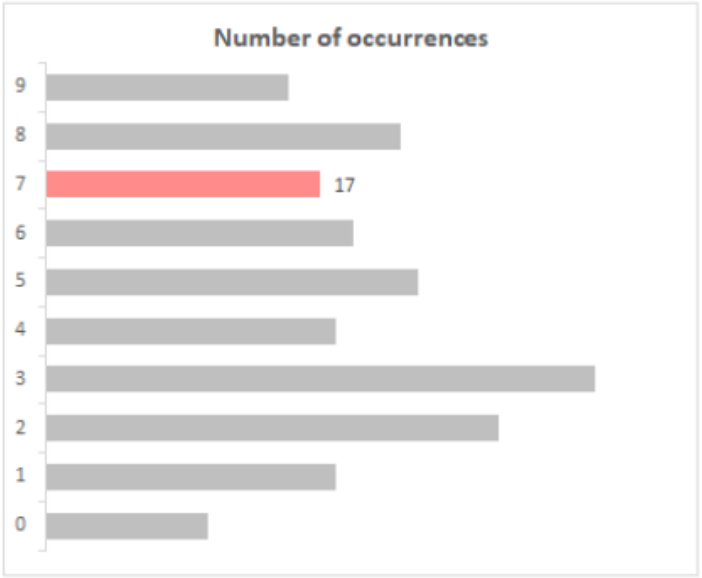
Activity

- Did you get them all?

5 2 8 3 6 1 9 3 6 2 5 3 **7** 4 3 8 3
8 5 8 9 6 2 1 4 4 3 9 3 6 5 2 4 9
1 0 2 **7** 5 2 8 3 6 1 6 2 9 3 8 3 8
5 8 4 **7** 2 0 3 **7** 3 5 4 **7** 1 8 2 0 1
2 5 3 6 4 3 9 1 0 8 9 5 **7** 3 4 5 3
2 **7** 5 2 8 3 6 1 6 2 9 3 8 3 8 5 8
4 **7** 2 0 3 **7** 3 5 4 **7** 1 8 2 0 1 9 6
2 1 4 4 3 9 3 6 5 2 4 9 1 0 2 **7** 5
2 8 3 6 1 6 2 9 3 8 3 8 5 8 4 **7** 2
0 3 **7** 3 5 4 **7** 1 8 2 0 1 2 5 3 6 4
3 9 1 8 9 5 0 **7** 3 4 5 3 2 **7** 5 2 8
3 6 1 6 2 4 6 2 **7** 5 9 1 5 2 6 3 6

Activity

- Is it easier to tell now?

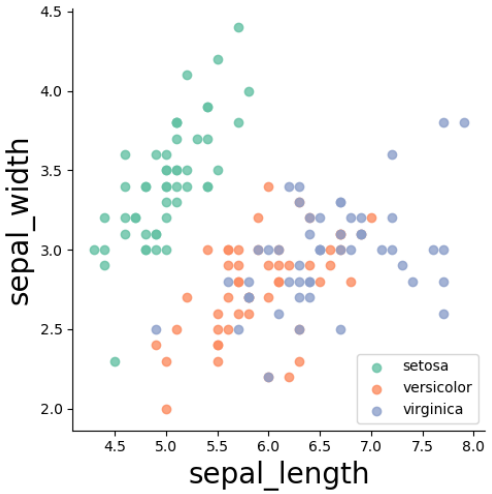


5	2	8	3	6	1	9	3	6	2	5	3	7	4	3	8	3
8	5	8	9	6	2	1	4	4	3	9	3	6	5	2	4	9
1	0	2	7	5	2	8	3	6	1	6	2	9	3	8	3	8
5	8	4	7	2	0	3	7	3	5	4	7	1	8	2	0	1
2	5	3	6	4	3	9	1	0	8	9	5	7	3	4	5	3
2	7	5	2	8	3	6	1	6	2	9	3	8	3	8	5	8
4	7	2	0	3	7	3	5	4	7	1	8	2	0	1	9	6
2	1	4	4	3	9	3	6	5	2	4	9	1	0	2	7	5
2	8	3	6	1	6	2	9	3	8	3	8	5	8	4	7	2
0	3	7	3	5	4	7	1	8	2	0	1	2	5	3	6	4
3	9	1	8	9	5	0	7	3	4	5	3	2	7	5	2	8
3	6	1	6	2	4	6	2	7	5	9	1	5	2	6	3	6

- In presenting data for analysis, visualization can make a big difference, just like it did for this activity

Visualizing data in Python

- In this course, we'll learn how to:
  - Make our data more readable and convincing using visualization
  - Generate basic visualizations/graphs using packages like matplotlib, plotly, cufflinks and explore advanced packages like bokeh to create interactive visuals
- But first...we begin where all data scientists do: **We will explore and learn more about the dataset** we will be using throughout the course and **clean and reshape it**



Course structure

- **Lecture slides:**
  - 2 hours

- **Knowledge checks and exercises:**
  - 30 minutes

Knowledge checks and exercises

- **Knowledge Checks:**
  - At the end of each mini-lesson, the instructor will launch a poll to check your knowledge
  - Read the questions carefully and answer each question

- **Exercises**
  - To complete the exercises, open the file named “Exercises Day 1” provided at the time of enrollment

Module completion checklist

Objective	Complete
Pre-work and introductions	
Introduce Costa Rican poverty dataset	
Load dataset and clean it using data cleaning techniques	
Reshape data using pandas	
Define use cases of Exploratory Data Analysis (EDA)	

For each module, we'll start out with our objectives for the session, so you know what to expect!

Module completion checklist

Objective	Complete
Pre-work and introductions	✔
Introduce Costa Rican poverty dataset	
Load dataset and clean it using data cleaning techniques	
Reshape data using pandas	
Define use cases of Exploratory Data Analysis (EDA)	

Costa Rican poverty: case study

- We will be diving into a case study from the **Inter-American Development Bank (IDB)**
- Recently, the **IDB** conducted a competition amongst data scientists on ***Kaggle.com***
- Many countries face the same problem of inaccurately assessing a social need
- The following case study on Costa Rican poverty levels is a good example of how we can use data science within social sciences

Costa Rican poverty: backstory

Costa Rican poverty level prediction

As stated by the **IDB**:

- Social programs have a hard time making sure the right people are given enough aid
- It's especially tricky when a program focuses on the poorest segment of the population
- The world's poorest typically can't provide the necessary income and expense records to prove that they qualify

Costa Rican poverty: backstory

Proxy Means Test (PMT)

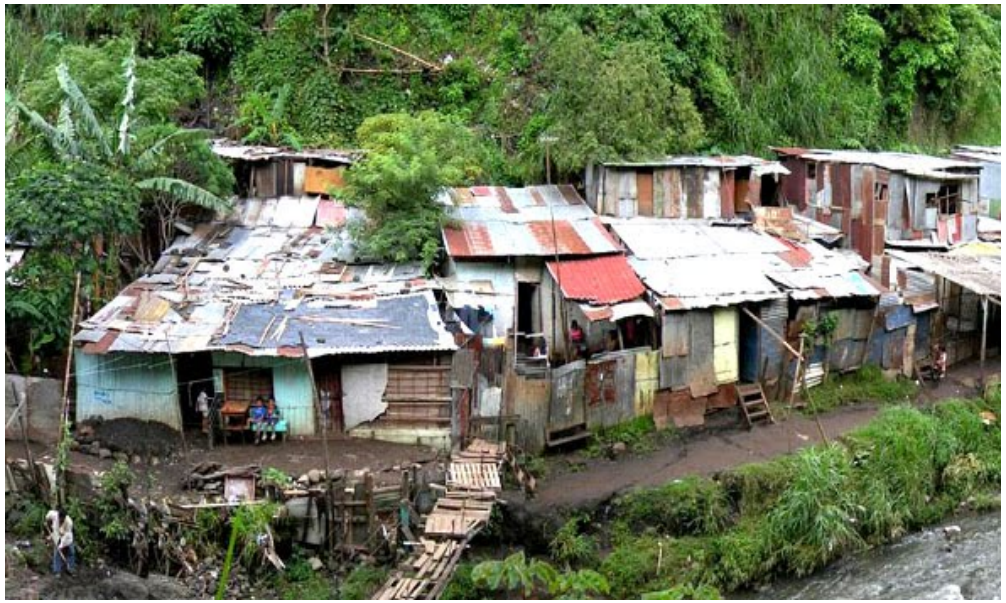
- In Latin America, one popular method uses an algorithm to verify income qualification: **Proxy Means Test (or PMT)**
- With the PMT, agencies use a model that considers a family's observable household attributes, such as the material of their walls and ceiling, or the assets found in the home, to classify them and predict their level of need
- While this is an improvement, accuracy remains a problem as the region's population grows and poverty declines

Costa Rican poverty: proposed solution

Costa Rican poverty level prediction: proposed solution

- To improve on PMT, the IDB built a competition for Kaggle participants to use methods beyond traditional econometrics

- The given dataset contains Costa Rican household characteristics with a target of four categories:
  - extreme poverty
  - moderate poverty
  - vulnerable households
  - non vulnerable households



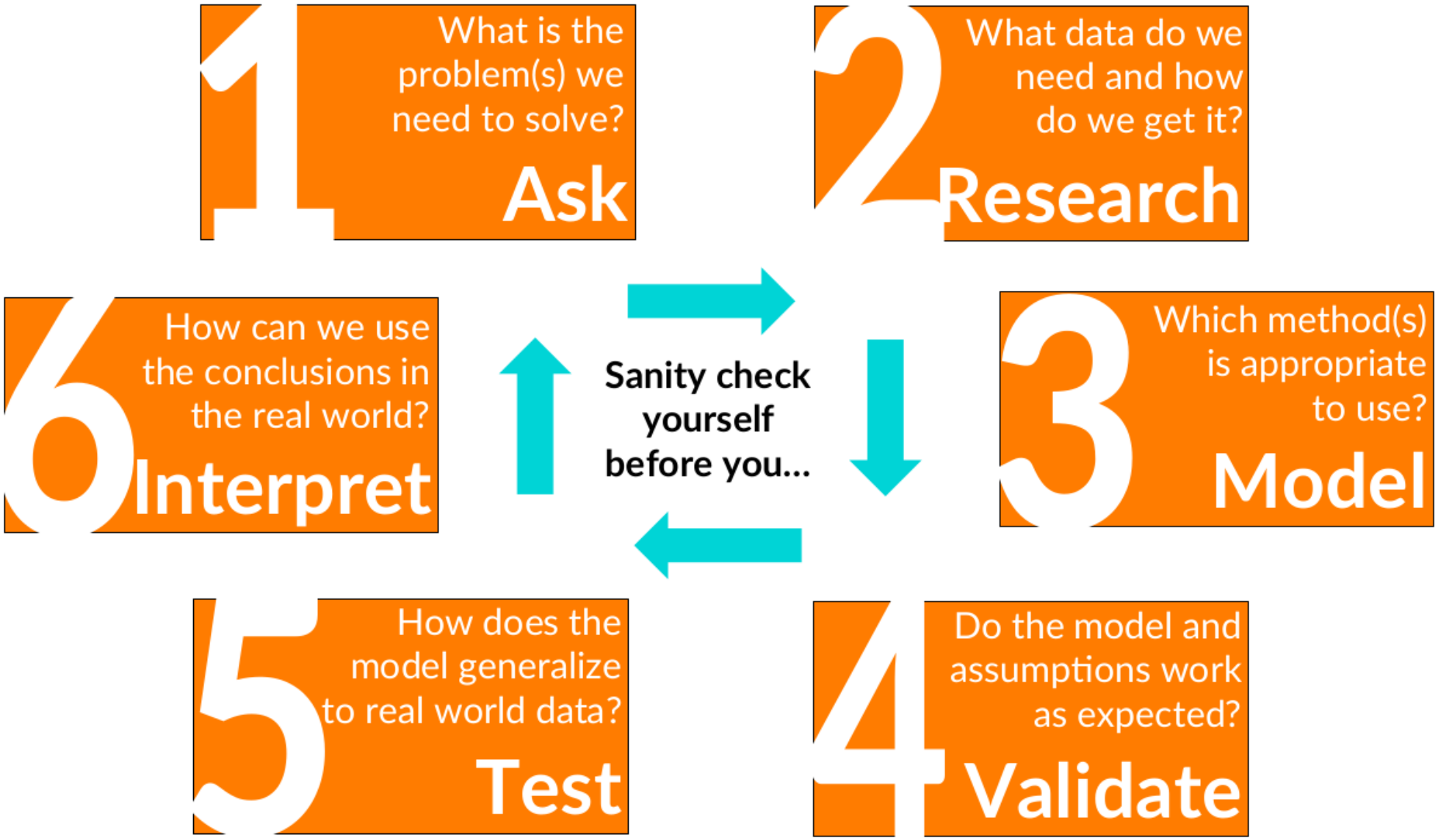
Costa Rican poverty: proposed solution

The goal is to develop an algorithm to predict these poverty levels, that can eventually be used not only on the Costa Rican population, but on other countries facing the same problem

We will work with the 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
- building static and interactive data visualizations

Data science control cycle



Module completion checklist

Objective	Complete
Pre-work and introductions	✔
Introduce Costa Rican poverty dataset	✔
Load dataset and clean it using data cleaning techniques	
Reshape data using pandas	
Define use cases of Exploratory Data Analysis (EDA)	

Loading packages

- Load the packages we will be using

```
import pandas as pd
import numpy as np
import pickle
import os
import matplotlib.pyplot as plt
```

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 `skillsoft-data-viz-with-python` folder

```
# Set `home_dir` to the root directory of your computer.
home_dir = os.path.expanduser("~")
# Set `main_dir` to the location of your `skillsoft-data-viz-with-python` folder.
main_dir = os.path.join(home_dir, "Desktop", "skillsoft-data-viz-with-python")
```

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

# Create a plot directory to save our plots
plot_dir = os.path.join(main_dir + "/plots")
```

Working directory

- Set working directory to `data_dir`

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

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

```
~/Desktop/skillsoft-data-viz-with-python/data
```

Data wrangling and exploration

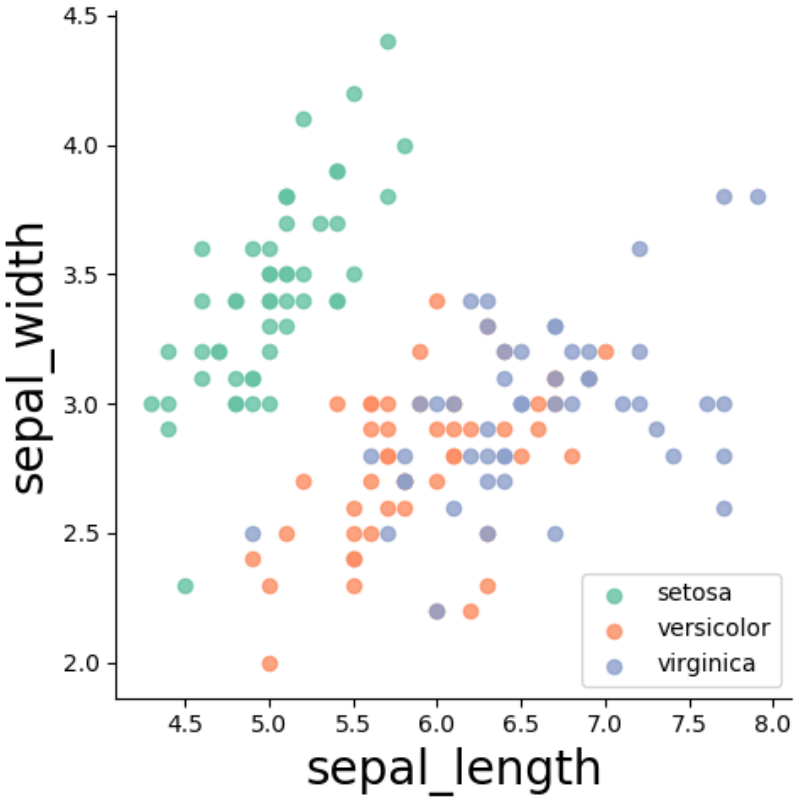
Remember, a data scientist must be able to:

1. **Wrangle** the data (gather, clean, and sample data to get a suitable data set)
2. **Manage** the data for easy access by the organization
3. **Explore** the data to generate a hypothesis

The techniques we will learn today will help us achieve these goals!

Why visualize data?

- Visual context provides **insights on patterns, trends and correlations** that might not be detected otherwise
- Simple way to **convey concepts** and provide visual access to **large amounts of complex data**
- Python has multiple graphing libraries with a lot of great features!



Costa Rica poverty dataset

We will load in our Costa Rican poverty dataset as `costa_rica_poverty`. This dataset includes information about:

- A target variable which represents **income level**, in which the **goal is to understand the relationship between the individual and household characteristics and the resulting income level**
- Individuals and households in the dataset are characterized by variables ranging from features about the house they live in, gender split of the household, education, region and a few other features
- Of the 84 characteristics, there are
  - **21 features** of the person or household's home
  - **26 features** of the gender split within the household and about the household
  - **15 features** of region and education
  - *22 other features* that also seem to be potential poverty level indicators about the household

Why this dataset?

- What are some key reasons this dataset is appropriate for the problem at hand?
- Think about the second step in our data science lifecycle
- Always make sure that you are working with the right data to answer the question

Load the dataset

- Let's load the entire dataset
- For visualizations, 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

- In this module, we will explore a subset of this dataset, which includes the following variables:
  - **household id**
  - **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 visualize and work with all of your data

Subsetting data

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

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

```
  household_id  ppl_total  dependency_rate  ...  age  monthly_rent  Target
0     21eb7fcc1          1               37  ...   43      190000.0       4
1     0e5d7a658          1               36  ...   67      135000.0       4
2     2c7317ea8          1               36  ...   92           NaN       4
3     2b58d945f          4               38  ...   17      180000.0       4
4     2b58d945f          4               38  ...   37      180000.0       4

[5 rows x 8 columns]
```

Remove labels

- We are now going to be working with the `costa_viz` that we just created

```
# Let's prepare the data for visualizations by removing any labels,
# removing the household_id variable, and keeping the remaining variables.
costa_viz = costa_viz.drop('household_id', axis = 1)
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(costa_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(costa_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 target 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
```

Pickle - what?

- We are going to pause for a moment to learn about the library `pickle`
- When we have objects we want to carry over and do not want to rerun code, we can `pickle` them
- In other words, `pickle` will help us **save objects** from one script/ session and **pull them up** in new scripts
- How do we do that? We use a function in Python called `pickle`. It is similar to **flattening** a file
  - **Pickle/saving:** a Python object is converted into a byte stream
  - **Unpickle/loading:** the inverse operation where a byte stream is converted back into an object

Pickle cleaned dataset

We'll now pickle our data set so it can be used for later

```
pickle.dump(costa_viz, open("costa_viz.sav","wb" ))
```

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Pre-work and introductions	✔
Introduce Costa Rican poverty dataset	✔
Load dataset and clean it using data cleaning techniques	✔
Reshape data using pandas	
Define use cases of Exploratory Data Analysis (EDA)	

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

- **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

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 some 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 Target
- We will group by the column `Target` for now

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

Summarizing using groupby()

- All the summary functions can be applied to a group
- For a refresher, 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

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)

```
# 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)
```

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

```
# 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

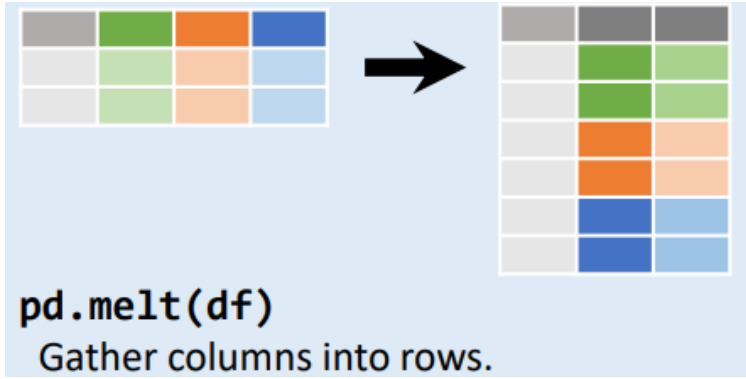
mrc_class	metric	mean
False	ppl_total	4.358607
True	ppl_total	3.796531
False	dependency_rate	26.011233
True	dependency_rate	25.425284
False	num_adults	2.388093
True	num_adults	2.713809
False	rooms	4.533839
True	rooms	5.205971
False	age	31.314238
True	age	36.078886

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,                              #<- 1
        id_vars = ['id_col'],    #<- 2
        var_name = 'some_var',   #<- 3
        value_name = 'some_val') #<- 4
```



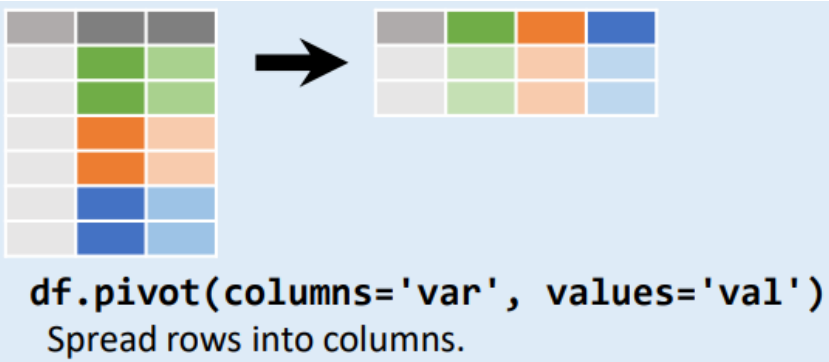
Wide to long format: melt (cont'd)

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

Note: *there is a slight difference in syntax between `melt`, which is a Pandas function, and `pivot`, which is a method of a dataframe. You would say `pd.melt()` but `df.pivot()` where `df` corresponds to any dataframe!*

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
```

Pickle grouped data frames

We'll now pickle our data set so it can be used for creating visualizations in next class

```
pickle.dump(costa_grouped_mean_long, open("costa_grouped_mean_long.sav","wb" ))
```

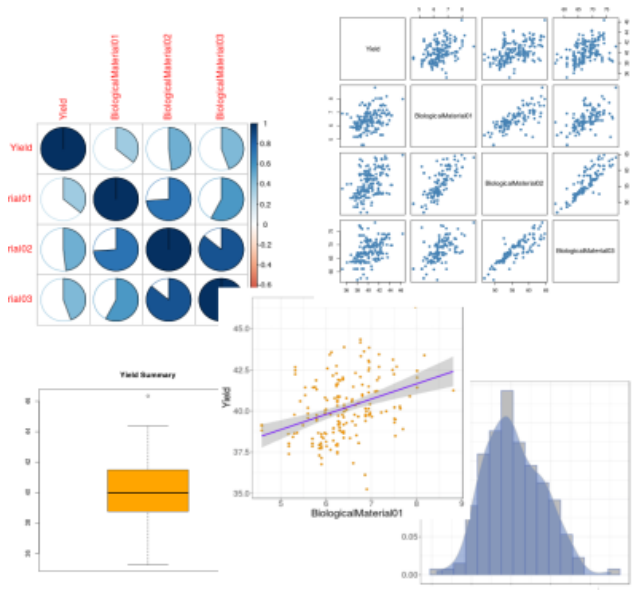
```
pickle.dump(costa_grouped_mean_wide, open("costa_grouped_mean_wide.sav","wb" ))
```

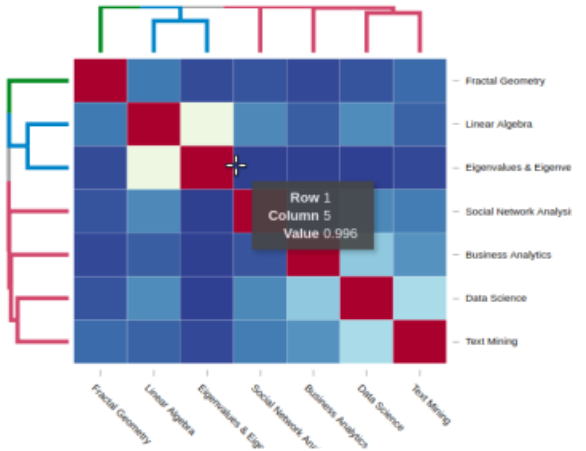
Module completion checklist

Objective	Complete
Pre-work and introductions	✔
Introduce Costa Rican poverty dataset	✔
Load dataset and clean it using data cleaning techniques	✔
Reshape data using pandas	✔
Define use cases of Exploratory Data Analysis (EDA)	

Why build a visualization?

- To provide valuable insights that are **interpretable** and **relevant**
- To give a visual or graphical representation of data / concepts
- To communicate ideas
- To provide an accessible way to see and understand trends, outliers, and patterns in data
- To confirm a hypothesis about the data



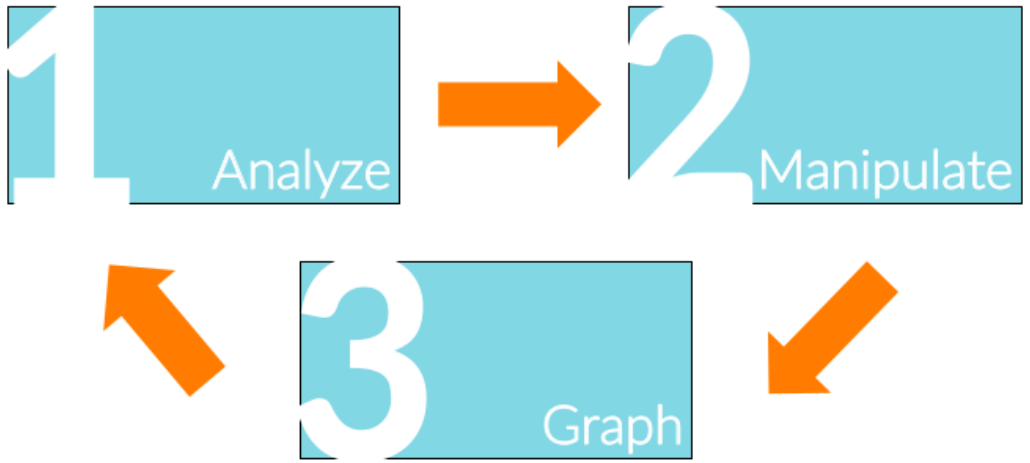


DATA SOCIETY © 2020

Exploratory data analysis

Python is a powerful tool for EDA because the graphics tie in with the functions used to analyze data. You can create graphs without breaking your train of thought as you explore your data. Visualization is an iterative process and consists of a few steps:

1. Analyze
2. Manipulate
3. Graph
4. Repeat



Exploratory data analysis in Python

What's possible

1. Visualization tools available through multitudes of packages (e.g. `matplotlib`, `seaborn`)
2. The visualizations created are high quality graphics that can be saved as `SVG`, `PNG`, `JPEG`, `BMP`, `PDF`
3. Visualizations are often the best way to display patterns in data for printed publications

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Pre-work and introductions	✔
Introduce Costa Rican poverty dataset	✔
Load dataset and clean it using data cleaning techniques	✔
Reshape data using pandas	✔
Define use cases of Exploratory Data Analysis (EDA)	✔

Summary

So far, we have explored:

1. Loading Costa Rican data set into R
2. Basic data cleaning techniques
3. Reshaping data using Pandas
4. Importance of visualizations and use cases of Exploratory data analysis

What we will cover in the next session:

1. Visualize Costa Rican poverty dataset by using `matplotlib` package
2. Understand univariate and bivariate plots
3. Customizing plots

This completes our module

Congratulations!