



{<coding:lab}

# Data Science 101

# Session Overview

{<coding:lab>}



- Introduction to Data Analytics
  - Data Analytics Process
- Pandas
  - DataFrame
  - Data Formats
  - Data Manipulation
  - Data Cleaning
- Plotly
  - Data Visualisation

{<coding:lab>}



# Introduction - What is Data Analytics?

# What Is Data Analytics?

{<coding:lab>}



- The process of extracting, processing and analysing data
- Deriving insights (useful information) from data
- Using the insights from data to support decision making
- Drawing a valid conclusion

# Who Uses Data Analytics?

{<coding:lab>}



- Commercial industries
  - Enable organizations to make more-informed business decisions
- Scientists and researchers
  - Verify or disprove scientific models, theories and hypotheses.

# Applications Of Data Analytics

{<coding:lab>}



- Retail Industry
  - Businesses can collect data about customer habits using data analytics to better understand their customers
  - This also helps them to predict trends and come up with new products to increase profit
- Pharmaceutical Industry
  - It is expensive to set up clinical trials
  - By using data analytics and artificial intelligence, we can improve the speed and efficiency of clinical trials

# Statistics For Data Analytics

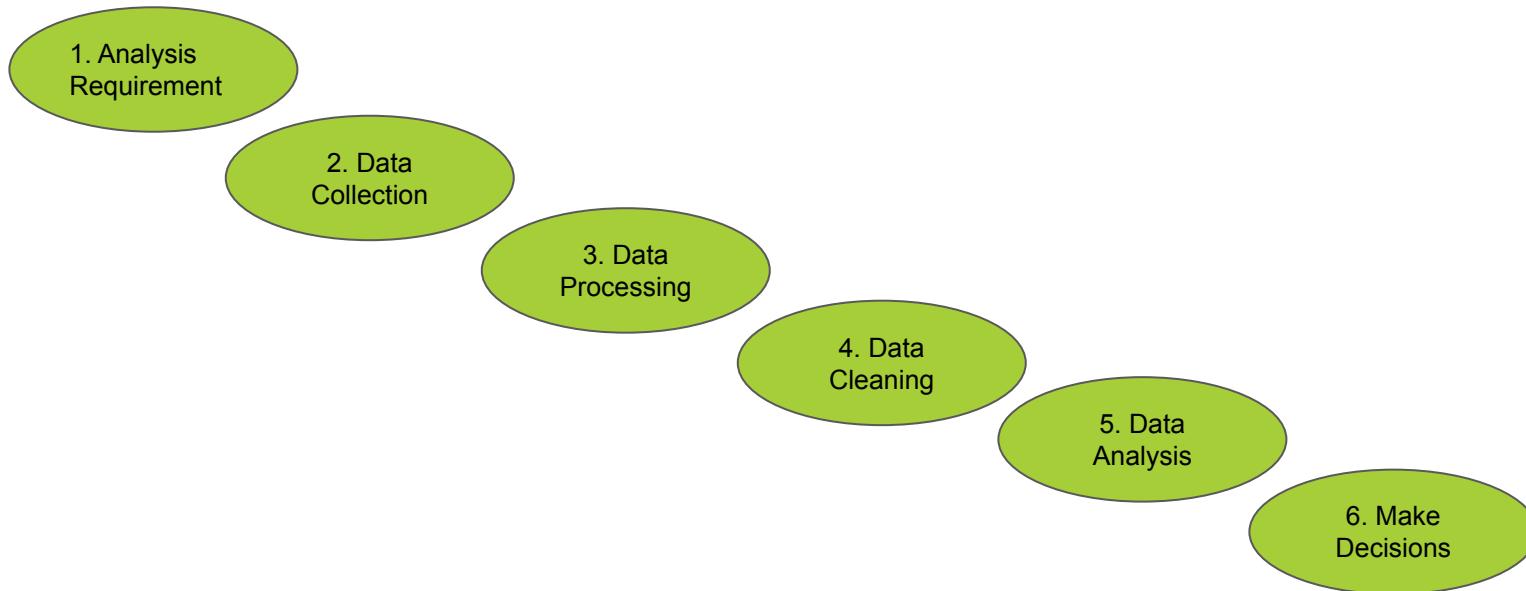
{<coding:lab>}



- Statistics is the science of learning from data
  - Data are numerical facts
- Includes methods of collecting, organizing and analyzing data
  - Such that meaningful conclusions can be drawn
- Basic statistics required as a skill set to do conduct analysis

# Data Analytics Process

{<coding:lab>}



# Data Analytics Process Explained {`coding:lab`}



- Define the analysis requirements
  - What is the business problem you want to solve?
- Data collection
  - The act of collecting and gathering the data
- Data processing
  - Putting together the collected data in a structured manner

# Data Analytics Process Explained {`coding:lab`}



- Data cleaning
  - Collected data is typically imperfect
  - We need to clean the data before analysis
- Data Analysis
  - Two key methods
    - Quantitative/Statistical Method
    - Graphical/Visualisation
- Make decisions
  - based on the conclusion from the data analysis

# What We Will Cover In This Session {`coding:lab`}



- Data processing
  - We will introduce to you a few common data format and Pandas has tools to read these format
- Data cleaning
  - Pandas has a few good tools for cleaning data
- Data Analysis
  - Focus on Graphical Methods for Visualisation

{<coding:lab>}



---

# Google Colab



# What Is Google Colab

{<coding:lab>}



- Google Colab is a free cloud service based on Jupyter Notebooks and it supports free GPU (Graphics Processing Unit)!
- Allows you to share your code with others
  - Others can view how you wrote your code and provide feedbacks
  - Similar to Google Doc and Google Drive
- We will be using Google Colab for most of our code in this workshop



- Often, we encounter errors on Python where it is unaware of which module you want

```
>>> import numpy as np
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
>>> |
```

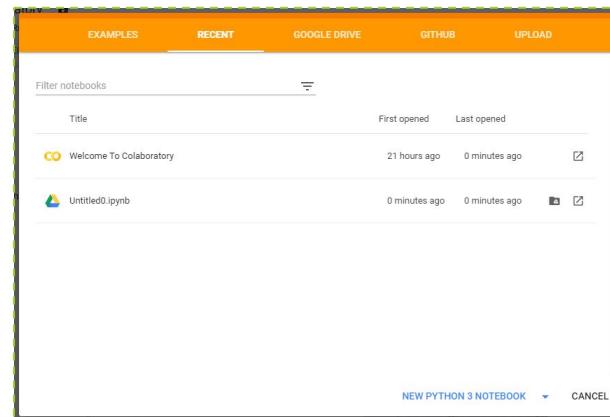
- Contains almost all the modules needed for data analysis

# Setting Up Google Colab

{<coding:lab>}



- Ensure you have signed in to your google account
- Start by opening a browser
  - <https://colab.research.google.com>
- Click on new Python 3 notebook

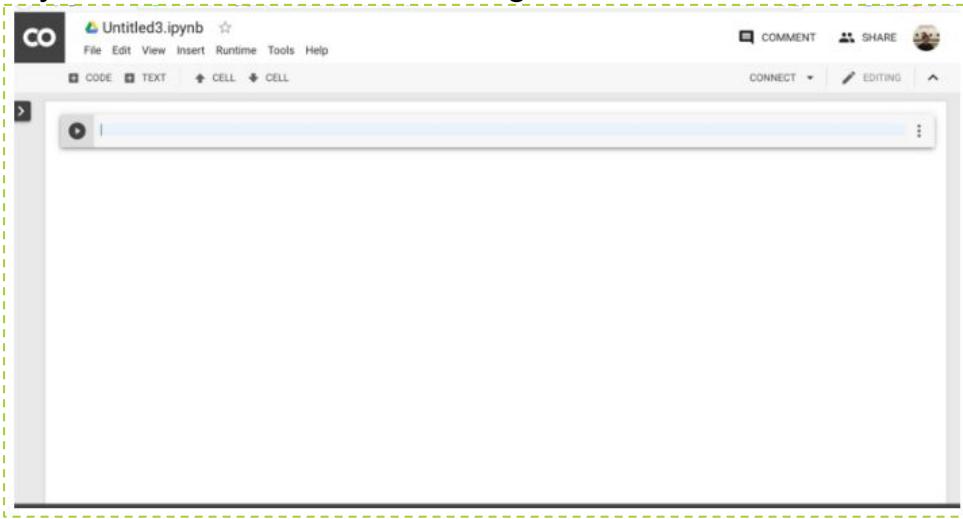


# Google Colab Layout

{<coding:lab>}



- By default, the surface of Google Colab looks similar to IDLE



# How Google Colab Works

{<coding:lab>}



- Runs cell by cell
- Shares your answer and provides a very useful function to find possible answers

```
[1] print "This is my solution!"  
      File "<ipython-input-1-a09954ca6674>", line 1  
          print "This is my solution!"  
                      ^  
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("This is my solution")?
```

SEARCH STACK OVERFLOW

# Sharing Codes

{<coding:lab>}



- Google Colab allows you to share your code with others
  - Similar to Google Docs
  - Permissions can also be selected
    - It is best to only allow others to view your code. Why?

# Google Colab (Demo/Practice - 1) {<coding:lab{}}



- If you have not already done so, create your first platform on google Colab
  - <https://colab.research.google.com>

# Checkpoint 1

{<coding:lab>}



- Every student must be able to:
  - Sign in to google account and opened a new Python Notebook on Google Colab
- For students who are waiting, try the following:
  - Familiarise yourself with the surface you would be working with

{<coding:lab>}



---

# Introduction to Pandas

# What Is Pandas?

{<coding:lab>}



- A software library written for the Python programming language for data analysis
  - Pandas stands for Python Data Analysis Library
- Offers data structures and operations for manipulating numerical tables

# Why Pandas?

{<coding:lab>}



- Pandas present data in a way that is suitable for data analysis
  - Series and DataFrame data structures
- Package contains multiple methods for convenient data filtering
- Pandas can read data from a variety of formats such as CSV, TSV, MS Excel, etc

# Pandas Installation

{<coding:lab>}



- Install the necessary library and import pandas into python
  - Since we are using Google Colab, Pandas library already exist

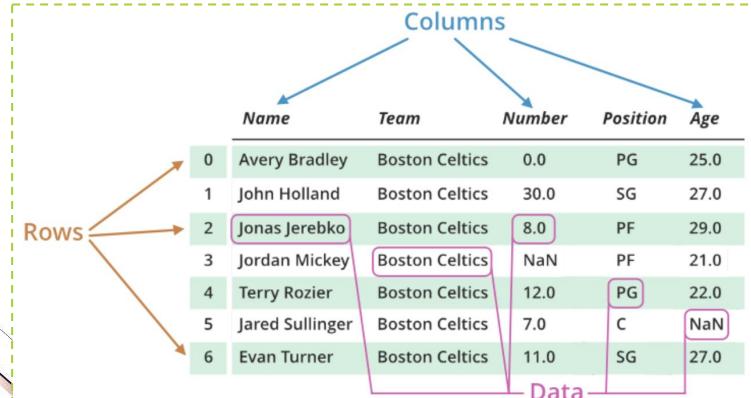
```
1 import pandas as pd
```

# DataFrame

{<coding:lab>}



- A two-dimensional, tabular data structure with labeled axes (rows and columns)
  - the primary pandas data structure
- You can think of a Dataframe as a spreadsheet in Python



# Creating DataFrames

{<coding:lab>}



- Multiple ways to create DataFrames using python
  - Manually
    - List of lists
    - List of Dictionaries
  - Updating an existing dataset
- All outputs would be presented neatly and beautifully in a table format

# List of Lists

{<coding:lab>}



- Creating multiple lists and storing them in a list
- Each column represents different categories

```
1 import pandas as pd
2 data = [
3     ['Alex', 20, 1050],
4     ['Bob', 52, 1400],
5     ['Cat', 23, 1690]
6 ]
7
8 df = pd.DataFrame(data, columns=['name', 'age', 'salary'])
9
10 print(df)
```

# List of Dictionaries

{<coding:lab>}



- Creating multiple dictionaries and storing them in a list
- What's the difference in terms of the output?

```
1 import pandas as pd
2
3 data = [
4     {'name': 'Alex', 'age': 20, 'salary': 1050},
5     {'name': 'Bob', 'age': 52, 'salary': 1400},
6     {'name': 'Cat', 'age': 23, 'salary': 1690}
7 ]
8
9 df = pd.DataFrame(data, columns=['name', 'age', 'salary'])
10
11 print(df)
```

# Series

{<coding:lab>}



- A series is a row or column of a DataFrame
- In another word, a DataFrame is made up of multiple Series stuck together vertically and horizontally

# Creating DataFrames (Demo/Practice Coding:lab)



- Try creating a DataFrame of your own (Name, weight, height)
  - List of Lists
  - List of Dictionaries

## Checkpoint 2

{<coding:lab>}



- Every student must be able to:
  - Understand what a DataFrame is and how to create one
  - Understand what a Series is
- For students who are waiting, try the following:
  - Read up if there are any other methods to create a DataFrame in Python
    - Which do you find easier?

{<coding:lab>}



---

# Structured Datasets

# What are Data Formats?

{<coding:lab>}



- A format in which data is stored
- The information is coded in such a way that a program or application can recognize, read and use the data

# Data Formats

{<coding:lab>}



- Common types of data formats
  - CSV
  - JSON
  - Excel
  - XML
- We will be learning how to work with CSV format in pandas
  - Other formats can be found in the Appendix

# Data Formats - Common Uses

{<coding:lab>}



| Data Format  | Common Uses   |
|--------------|---|
| CSV          | Simple file format used to store tabular data, such as a spreadsheet or database  |
| JSON         | Primarily used to transmit data between a server and web applications   |
| XML          | XML is used to store or transport data in HTML applications. HTML is used to format and display the same data. XML separates the data from HTML |
| Excel (.xls) | Spreadsheet file created by Microsoft Excel   |

# CSV File Format

{<coding:lab>}



- The most common format is a Comma Separated Value (CSV) format
  - Most people know how it works and
  - Easily viewable in various products
    - Including Microsoft Excel and Python
- A CSV file is a human readable text file where each line has a number of fields, separated by commas or some other delimiter

# CSV files on Google Colab

{<coding:lab>}



- In order to read files on Google Colab, we have to import them first (Recommend using Google Chrome!)

```
1 #importing csv files onto google colab
2 from google.colab import files
3 uploaded = files.upload()
```

- Successfully uploaded files would look like this

```
[1] #importing csv files onto google colab
from google.colab import files
uploaded = files.upload()

 data.csv
• data.csv(application/vnd.ms-excel) - 146357 bytes, last modified: 6/16/2019 - 100% done
Saving data.csv to data.csv
```

# Using Pandas to Read CSV

{<coding:lab>}



- We can read from CSV using **read\_csv()** function
  - First import the file onto Google Colab
  - Return the file into Dataframe format

```
1 import pandas as pd  
2  
3 pd.read_csv('unit05-data.csv')
```

# Extracting data in DataFrame

{<coding:lab>}



- We can extract columns and rows from a Data Frame
  - Exact syntax will be covered later
- For example, I want to find out the total number of cases of TB per 100k people in Singapore from 2000 to 2016
  - Simply extract the row where the country is Singapore and add them up using `.sum()`

```
1 import pandas as pd
2 df =pd.read_csv('unit05-data.csv')
3
4 sg = df[ (df['country'] == 'Singapore')]
5 sg['Estimated incidence (all forms) per 100 000 population'].sum()
```

# Reading CSV in DataFrame (Demo/Practicing 3) ab }



- unit05-data.csv is a csv file which contains data regarding the cases of Tuberculosis in multiple countries
- Using the file unit05-data.csv
  - How many cases of Tuberculosis were there in 2008 in total?
  - Find out how many cases of Tuberculosis per 100k people there were in Zimbabwe from 2000 to 2016

## Checkpoint 3

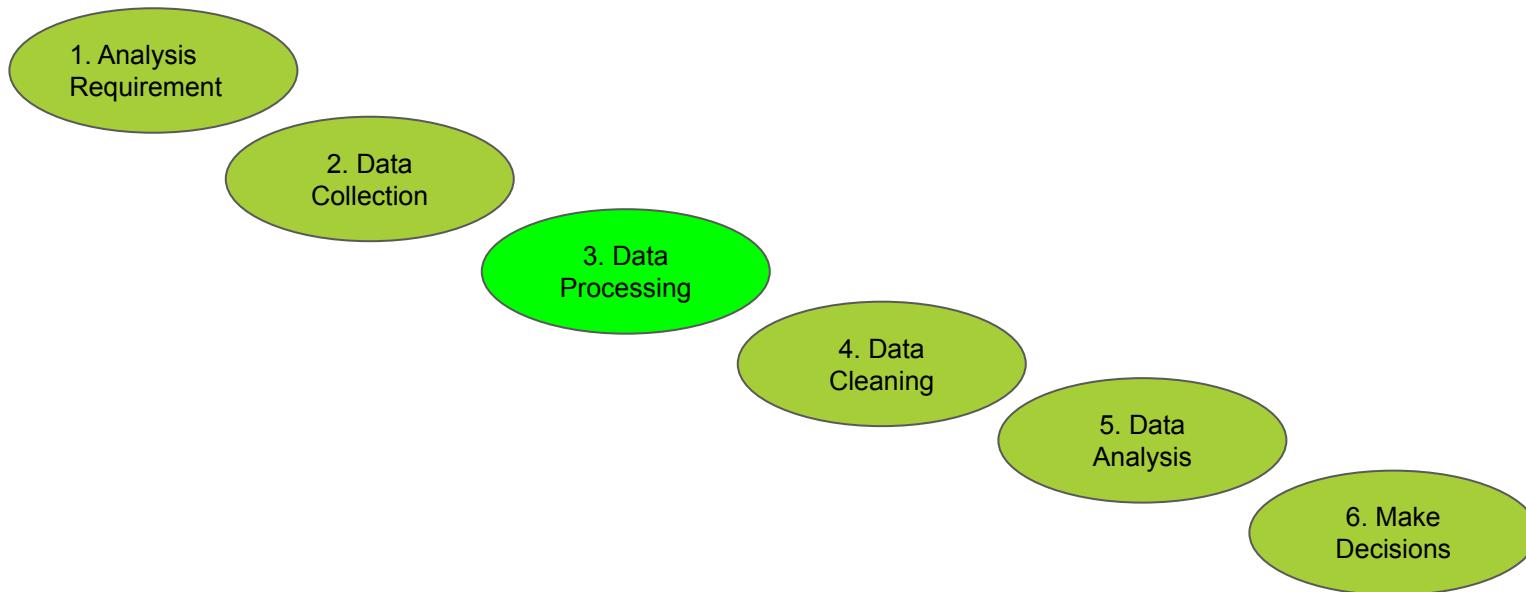
{<coding:lab>}



- Every student must be able to:
  - Open csv files into Dataframe
  - Extract specific data from dataframe
- For students who are waiting, try the following:
  - Access the csv file and add up the total number of Tuberculosis for each country
  - Read up on other ways to open csv files
    - Hint: Search up csv.writer()

# Data Analytics Process

{<coding:lab>}



# Manipulating Data for Quantitative Analysis

{Analysing:lab}



- Common Data Manipulation Methods
  - Extract column, Extract row
  - Grouping and Aggregation - By single column, by multiple columns
  - Filtering (selecting rows based on 1 or more criteria)
  - Joining related data from different tables
  - Pivoting
- Application of Statistical Methods will be covered in Session 3

# Extracting A Row From DataFrame {coding:lab}



- To extract a row, we use `loc()` function with the desired index we want

```
1  data = [
2      {'name': 'Cat', 'age': 23, 'salary': 1690},
3      {'name': 'Alex', 'age': 20, 'salary': 1050},
4      {'name': 'Bob', 'age': 52, 'salary': 1400}
5
6  ]
7
8  df = pd.DataFrame(data, columns=['name', 'age', 'salary'])
9  print(df.loc[0])
```

# Extracting A Column From DataFrame



- On the other hand, to extract a column, we simply type in the title of the column we want

```
1 data = [
2     {'name': 'Cat', 'age': 23, 'salary': 1690},
3     {'name': 'Alex', 'age': 20, 'salary': 1050},
4     {'name': 'Bob', 'age': 52, 'salary': 1400}
5
6 ]
7
8 df = pd.DataFrame(data, columns=['name', 'age', 'salary'])
9 print(df['name'])
```

# Extracting A Row And Column (Demo/Practice)

{Coding Tab} 4



- Based on the DataFrame you have created earlier
  - Extract a row
  - Extract a column

## Checkpoint 4

{<coding:lab>}



- Every student must be able to:
  - Extract a particular row or column from DataFrame
  
- For students who are waiting, try the following:
  - What are other functions or methods you have discovered that can extract rows and columns?

# Grouping and Aggregation in Pandas



- Pandas comes with a whole host of sql-like aggregation functions you can apply when grouping on one or more column

# Sample Dataset

{<coding:lab>}



- Let's start by creating a sample dataset

```
1 import pandas as pd
2 student = ["Anna", "Bob", "Cass", "Darius", "Eugene", "Felica", "Gordon"]
3 classroom = ["A", "A", "B", "A", "B", "B", "B"]
4 age = [15, 15, 16, 17, 15, 18, 15]
5 score = [80, 82, 95, 70, 67, 98, 85]
6 data = {"Student": student, "Class": classroom, "Age": age, "Score": score}
7 df = pd.DataFrame(data)
8 df
```

# Grouping and Aggregation Example {`coding:lab`}



- First we'll group by Class with Pandas' **groupby** function
- After grouping we can pass **aggregation** functions to the grouped object as a dictionary within the `agg` function
  - This dict takes the column that you're aggregating as a key, and either a single aggregation function or a list of aggregation functions as its value.
  - To apply aggregations to multiple columns, just add additional key:value pairs to the dictionary.

# Grouping and Aggregation Code

{<coding:lab>}



- Sample code:

```
1 import pandas as pd
2 student = ["Anna", "Bob", "Cass", "Darius", "Eugene", "Felica", "Gordon"]
3 classroom = ["A", "A", "B", "A", "B", "B", "B"]
4 age = [15, 15, 16, 17, 15, 18, 15]
5 score = [80, 82, 95, 70, 67, 98, 85]
6 data = {"Student": student, "Class": classroom, "Age": age, "Score": score}
7 df = pd.DataFrame(data)
8 class_score = df.groupby('Class').agg({'Score': ['mean', 'min', 'max']})
9 class_score
```

# Grouping by Multiple Columns

{<code>:lab}



- It's simple to extend this to work with multiple grouping variables.
- Say we want to summarise student score by class AND age.
  - We can do this by passing a list of column names to `class_score` instead of a single string value

# Grouping by Multiple Columns: Code#9ACD32">coding:lab }



- Sample Code:

```
1 import pandas as pd
2 student = ["Anna", "Bob", "Cass", "Darius", "Eugene", "Felica", "Gordon"]
3 classroom = ["A", "A", "B", "A", "B", "B", "B"]
4 age = [15, 15, 16, 17, 15, 18, 15]
5 score = [80, 82, 95, 70, 67, 98, 85]
6 data = {"Student": student, "Class": classroom, "Age": age, "Score": score}
7 df = pd.DataFrame(data)
8 class_score = df.groupby(['Class', 'Age']).agg({'Score': ['mean', 'min', 'max']})
9 class_score
```

# Grouping & Aggregation (Demo/Practice - 5)

{<coding:lab>}



- Create your own sample dataset
  - Similar to the earlier example
- Apply grouping and aggregation to your dataset

## Checkpoint 5

{<coding:lab>}



- Every student must be able to:
  - Create a sample dataset
  - Group and aggregate the data
- For those who are waiting, try the following:
  - Try performing grouping and aggregation on one of the bigger datasets given (e.g. Unit05-Demo53.xls)
    - Summarise the ages by gender and country

# Filtering

{<coding:lab>}



- One of the biggest advantages of having the data as a Pandas Dataframe is that Pandas allows us to slice and dice the data in multiple ways
- Often, we may want to subset a pandas dataframe based on one or more values of a specific column
  - Essentially, we would like to select rows based on one value or multiple values present in a column

# Filtering Scenario

{<coding:lab>}



- Let's go back to the tuberculosis example
  - Huge dataframe with 3651 rows and 7 columns

# Select Rows Based on Value of Column



- One way to filter by rows in Pandas is to use boolean expression
- We first create a boolean variable by taking the column of interest and checking if its value equals to the specific value that we want to select/keep

## Select Rows: Code

{<coding:lab>}



- For example, let us filter the dataframe or subset the dataframe based on year's value 2002
  - This conditional results in a boolean variable that has True when the value of year equals 2002, False otherwise
  - We can then use this boolean variable to filter the dataframe.

```
1 import pandas as pd
2 df = pd.read_csv('unit05-data.csv')
3
4 year_2002 = df[ (df['year'] == 2002)]
5 year_2002
```

# Selection Results

{<coding:lab>}



- After subsetting we can see that new dataframe is much smaller in size

|      | country                   | iso2 | iso3 | Region | Code | year | Total Population | Estimated incidence (all forms) per 100 000 population |
|------|---------------------------|------|------|--------|------|------|------------------|--|
| 2    | Afghanistan               | AF   | AFG  | EMR    | 2002 |      | 21979923         | 189.0  |
| 19   | Albania                   | AL   | ALB  | EUR    | 2002 |      | 3119029          | 22.0   |
| 36   | Algeria                   | DZ   | DZA  | AFR    | 2002 |      | 31995046         | 74.0   |
| 53   | American Samoa            | AS   | ASM  | WPR    | 2002 |      | 58731            | 3.9  |
| 70   | Andorra                   | AD   | AND  | EUR    | 2002 |      | 70049            | 8.2  |
| ...  | ...                       | ...  | ...  | ...    | ...  |      | ...              | ...  |
| 3568 | Wallis and Futuna Islands | WF   | WLF  | WPR    | 2002 |      | 14622            | 149.0  |
| 3585 | West Bank and Gaza Strip  | PS   | PSE  | EMR    | 2002 |      | 3379049          | 1.9  |
| 3602 | Yemen                     | YE   | YEM  | EMR    | 2002 |      | 18919179         | 102.0  |
| 3619 | Zambia                    | ZM   | ZMB  | AFR    | 2002 |      | 11120409         | 695.0  |
| 3636 | Zimbabwe                  | ZW   | ZWE  | AFR    | 2002 |      | 12500525         | 617.0  |

213 rows × 7 columns

# Select Rows Whose Column Value Does Not Equal a Specific Value

{<code>:lab}



- Sometimes, you may want to keep rows of a data frame based on values of a column that does not equal something
- Let us filter our TB dataframe whose year column is not equal to 2002
  - Basically we want to have all the years' data except for the year 2002.

## Select Not Equal Rows: Code

{<coding:lab>}



- Instead of the equivalence operator (==), we can now use the non-equivalence operator (!=)

```
1 import pandas as pd
2 df = pd.read_csv('unit05-data.csv')
3
4 year_not_2002 = df[ (df['year'] != 2002)]
5 year_not_2002
```

# Select Rows Whose Column Value is Not NA/NAN {coding:lab}



- Often we may want to filter a Pandas dataframe such that we keep the rows if values of certain column is not NA/NAN.
- We can use Pandas `notnull()` method to filter based on NA/NAN values of a column.

```
1 import pandas as pd
2 df = pd.read_csv('unit05-data.csv')
3 df_not_null = df[df.year.notnull()]
4 df_not_null
```

# Pandas isin()

{<coding:lab>}



- Pandas DataFrame.**isin()** function allows us to select rows using a list or any iterable
- If we use isin() with a single column, it will simply result in a boolean variable with True if the value matches and False if it does not

# Select Rows Based on a List

{<coding:lab>}



- Sometimes, we want to select rows using multiple values present in an iterable or a list
  - For example, let us say we want select rows for years [2002, 2008]

```
1 import pandas as pd
2 df = pd.read_csv('unit05-data.csv')
3
4 years = [2002, 2008]
5 selection = df[df.year.isin(years)]
6
7 selection
```

## Negation Symbol (~)

{<coding:lab>}



- We can also select rows based on values of a column that are not in a list or any iterable
- We will create a boolean variable just like before, but now we will negate the boolean variable by placing ~ in the front

# Select Rows Based on Values Not in List



- For example, to get rows where the year is not 2002 nor 2008, we can use the following code:

```
1 import pandas as pd
2 df = pd.read_csv('unit05-data.csv')
3
4 years = [2002,2008]
5 selection = df[~df.year.isin(years)]
6
7 selection
```

# Select Rows Using Multiple Conditions



- We can combine multiple conditions using & operator
  - For example, we can combine the above two conditions to get Singapore's data from years 2002 and 2008

```
1 import pandas as pd
2 df = pd.read_csv('unit05-data.csv')
3
4 years = [2002,2008]
5 country = ["Singapore"]
6 selection = df[df.year.isin(years) & df.country.isin(country)]
7
8 selection
```

# Filtering (Demo/Practice - 6)

{<coding:lab>}



- Using unit05-data.csv, try the following
  - Select Singapore's data from years 2002 and 2008
  - Select data between 2004 and 2008 that are not from Singapore

## Checkpoint 6

{<coding:lab>}



- Every student must be able to:
  - Filter and select rows of a DataFrame
- For those who are waiting, try the following:
  - Try filtering through the DataFrame for other countries' information, as well as information from other years

# Joining Data From Multiple Data Frames



- In many scenarios, our data are often disjointed
  - Meaning they come in multiple files
- In order to process the data, we will first need to join the dataframes

# Creating More Datasets

{<coding:lab>}



```
1 import pandas as pd
2 student1 = ["Anna", "Bob", "Cass", "Darius", "Eugene", "Felica", "Gordon"]
3 classroom1 = ["A", "A", "B", "A", "B", "B", "B"]
4 age1 = [15, 15, 16, 17, 15, 18, 15]
5 score1 = [80, 82, 95, 70, 67, 98, 85]
6 data1 = {"Student": student1, "Class":classroom1, "Age": age1,"Score": score1}
7 df1 = pd.DataFrame(data1)
8
9 student2 = ["Charlie", "James", "Ethan"]
10 classroom2 = ["C", "C", "C"]
11 age2 = [16, 17, 18]
12 score2 = [95, 77, 80]
13 data2 = {"Student": student2, "Class":classroom2, "Age": age2,"Score": score2}
14 df2 = pd.DataFrame(data2)
```

# Concatenate By Row and Column {`<coding:lab>`}



- We can concatenate two dataframes by using `pd.concat()`
  - What happens if we leave out `ignore_index = True`?

```
16 df_row = pd.concat([df1, df2], ignore_index=True)
```

- To (17) df\_row  
  - But this will not make much sense in our example here
  - Will be useful if we wish to add in more fields for each student
    - E.g. Parents' name

# Merge DataFrames

{<coding:lab>}



- Two DataFrames might hold different kinds of information about the same entity and linked by some common feature/column
- To join these DataFrames, pandas provides multiple functions like **concat()**, **merge()** , **join()**, etc
- We will cover **merge()**

# Merge DataFrames on Specific Keys



- Say we want to add in the students' id numbers
  - We have a separate data frame holding the id numbers
  - We can use `pd.merge()` to merge the data frames

```
18 student3 = ["Anna", "Bob", "Cass", "Darius", "Eugene", "Felicia", "Gordon", "Charlie", "James", "Ethan"]
19 id3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
20 data3 = {"Student":student3, "id":id3}
21 df3 = pd.DataFrame(data3,columns = ['Student', 'id'])
22
23 df_merge_col = pd.merge(df_row, df3, on="Student")
24 df_merge_col
```

# Merge Dataframes (Demo/Practice) { ding:lab }



- Create 2 more datasets to supplement the dataset you created earlier
  - Create dataframes and try to merge your datasets
  - What happens when the order of the names are different?
  - How about when there is a different name?

## Checkpoint 7

{<coding:lab>}



- Every student must be able to:
  - Concatenate and Merge data frames
- For those who are waiting, try the following:
  - Read up on pd.concat
    - What other arguments can it accept?

# Pivot Tables

{<coding:lab>}



- A pivot table is a table of statistics that summarizes the data of a more extensive table
- A pivot table calculates a statistic on a breakdown of values
  - For the first column, it displays values as rows and for the second column as columns

# Pivoting in Pandas

{<coding:lab>}



- Pandas has a **pivot\_table** function that applies a pivot on a DataFrame.
- It also supports aggfunc that defines the statistic to calculate when pivoting
  - aggfunc is np.mean by default, which calculates the average

# Pivoting in Pandas: Example

{<coding:lab>}



- Going back to our earlier dataset we used for merging, suppose we would like to have a breakdown of average scores with class in rows (specify index) and age in columns (specify columns)

```
27 df_merge_col.pivot_table(index="Class", columns="Age", values="Score")
```

# Pivoting in Pandas: Result

{<coding:lab>}



- The pivot table should look something like this

| Age   | 15   | 16   | 17   | 18   |
|-------|------|------|------|------|
| Class |      |      |      |      |
| A     | 81.0 | NaN  | 70.0 | NaN  |
| B     | 76.0 | 95.0 | NaN  | 98.0 |
| C     | NaN  | 95.0 | 77.0 | 80.0 |

# Set Missing Values To 0

{<coding:lab>}



- pivot\_table has a **fill\_value** argument to replace missing values with
  - None by default

```
|27 df_merge_col.pivot_table(index="Class", columns="Age", values="Score", fill_value = 0)
```

# Pandas Pivot Table (Demo/Practice{-8)ding:lab }



- Create a Pivot Table
  - Calculate the maximum and minimum score by class ages
  - Set missing values to 0

## Checkpoint 8

{<coding:lab>}



- Every student must be able to:
  - Create a Pivot Table
- For those who are waiting, try the following:
  - Read up on Pivot Tables
    - What is the difference between pd.pivot() and pd.pivot\_table()

# Outputting The Data

{<coding:lab>}

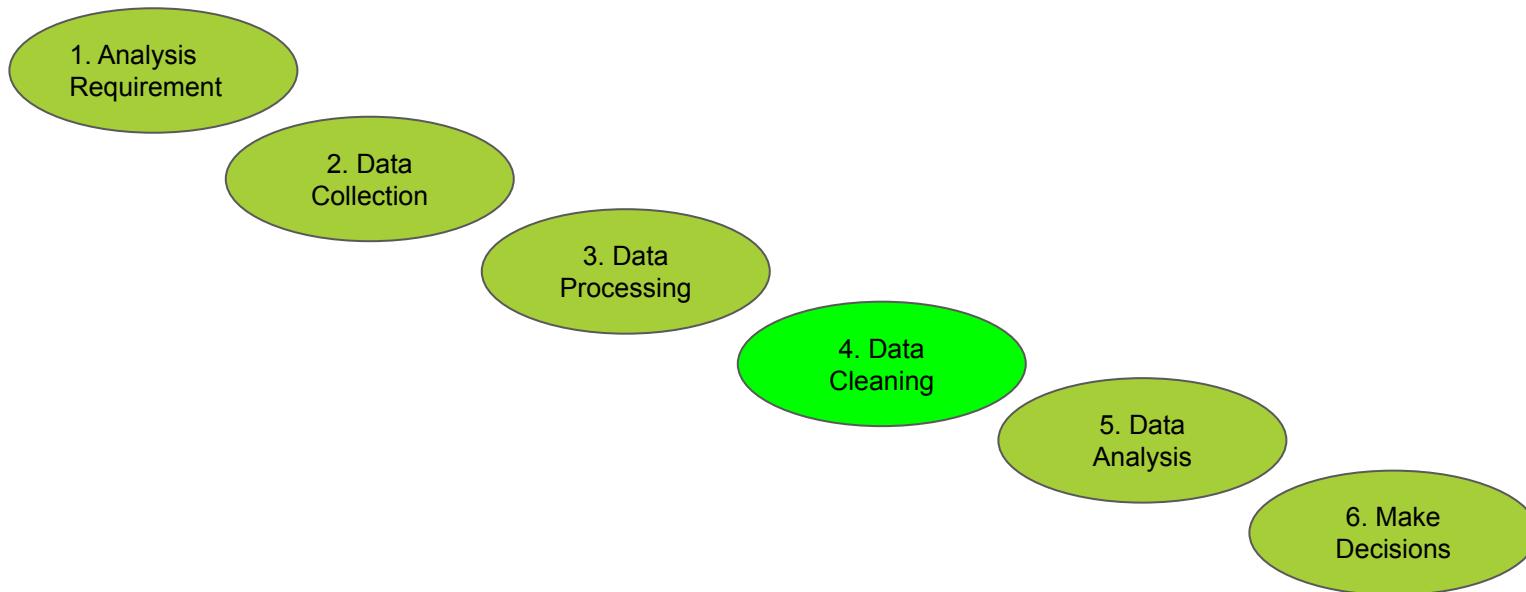


- Now that we have our pivot table, let's output it into a .csv file

```
1 df_merge_col.to_csv (r'df_merge_col.csv', index = False, header=True)
```

# Data Analytics Process

{<coding:lab>}



# Unstructured Data

{<coding:lab>}



- Our dataset usually contains different types of data in different formats
  - But a machine can only understand mathematical data.
- Our data may contain colors, date, gender, city etc. which are not in numerical format.
  - So they required to be converted in some numerical formats.

# What is Data Preprocessing?

{<coding:lab>}



- Sometimes data values are in different ranges which can cause some problems in learning.
  - So they need to be converted into some fixed ranges.
- Data preprocessing is simply converting these types of data into some valid numerical form in order for us to successfully perform data analysis
- Data preprocessing plays a very important role in Data Analytics

# Data Preprocessing in Pandas

{<coding:lab>}



- Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data
  - Load
  - Prepare
  - Manipulate
  - Model
  - Analyze

# Missing Data

{<coding:lab>}



- There are some specific rows in our dataset which doesn't contain any information for some specific columns.
- Handling missing values is very common in Data Analytics as our programs often do not work if there are missing rows or columns in a data frame

# Handling Missing Data

{<coding:lab>}



- Missing data can be handled commonly in two ways
  - Removing data
  - Imputation

# Removing Data

{<coding:lab>}



- In many cases, the solution is just removing the specific row. If we use pandas, then it is very simple. We just need to use one pandas methods called **dropna()**
- Suppose we have a panda dataframe df, which contains some missing values.

# Removing Data Example

{<coding:lab>}



- Let's look at "MissingData.xls"
  - The file contains the information of a group of people who visited a restaurant on a particular date
  - A few ages are omitted, say, because some customers did not want to reveal their age
  - These customers did not get the discount
- The restaurant had a promotion on 21 May 2015, where customers 25 and above get to dine in for free
  - Find out the total number of people who enjoyed this perk

# New Table With Missing Data

{<coding:lab>}



- Our table now looks something like this

|    | A  | B          | C         | D      | E             | F   | G          | H    |
|----|----|------------|-----------|--------|---------------|-----|------------|------|
| 1  | 0  | First Name | Last Name | Gender | Country       | Age | Date       | Id   |
| 2  | 1  | Dulce      | Abril     | Female | United States | 32  | 15/10/2017 | 1562 |
| 3  | 2  | Mara       | Hashimoto | Female | Great Britain | 25  | 16/08/2016 | 1582 |
| 4  | 3  | Philip     | Gent      | Male   | France        |     | 21/05/2015 | 2587 |
| 5  | 4  | Kathleen   | Hanner    | Female | United States |     | 15/10/2017 | 3549 |
| 6  | 5  | Nereida    | Magwood   | Female | United States |     | 16/08/2016 | 2468 |
| 7  | 6  | Gaston     | Brumm     | Male   | United States | 24  | 21/05/2015 | 2554 |
| 8  | 7  | Etta       | Hurn      | Female | Great Britain | 56  | 15/10/2017 | 3598 |
| 9  | 8  | Earlean    | Melgar    | Female | United States | 27  | 16/08/2016 | 2456 |
| 10 | 9  | Vincenza   | Weiland   | Female | United States | 40  | 21/05/2015 | 6548 |
| 11 | 10 | Fallon     | Winward   | Female | Great Britain | 28  | 16/08/2016 | 5486 |
| 12 |    |            |           |        |               |     |            |      |

# Remove Row with Missing Data

{<coding:lab>}



- We can use the **dropna()** function
  - **dropna()** takes in 1 optional argument, **axis**, which determines whether we remove a row or column. By default, it is 0 for row

```
1 import pandas as pd
2 df = pd.read_excel("MissingData.xls", typ = "series")
3 df = df.dropna()
4 df
```

# Remove Column with Missing Data {coding:lab}



- If we want to remove the column instead, we can set the argument axis = 1 inside the dropna() method
  - But in this example, it makes more sense to remove the row rather than the column

```
1 import pandas as pd  
2 df = pd.read_excel("MissingData.xls", typ = "series")  
3 df = df.dropna(axis=1)  
4 df
```

# Missing Data (Demo/Practice - 9) {`coding:lab`}



- Using “MissingData.xls”, try the following:
  - Remove rows with missing data
  - Remove columns with missing data

## Checkpoint 9

{<coding:lab>}



- Every student must be able to:
  - Remove rows with missing data
  - Remove columns with missing data
- For those who are waiting, try the following:
  - How would removing rows/column with missing data affect our overall dataset?

# Removing Data?

{<coding:lab>}



- Removing data, while convenient, is usually not the best solution.
- Sometimes deleting a row can delete some important information of other columns
- So there is another method called 'Imputation'

# Imputation

{<coding:lab>}



- In imputation, instead of deleting a row, we fill the missing values by some other values
- The imputed values might be median, mean, 0 etc.
- The imputed values might not be exactly the same but it is very accurate to the right value

# Imputation Scenario

{<coding:lab>}



- We earlier established that customers who did not reveal their age were not eligible for the discount
  - Hence we do not need to count them in
  - But we might not want to remove the row entirely, as removing them will affect things like the overall percentage of customers who were given discounts

# Imputation in Pandas

{<coding:lab>}



- Since we know that only customers who are 25 and above will get a free meal, we can set the unknown to 0 so they will not be counted

```
1 import pandas as pd  
2 df = pd.read_excel("MissingData.xls")  
3 df["Age"] = df["Age"].fillna(0)  
4 df
```

# Imputation (Demo/Practice - 10) {<coding:lab>}



- Using “MissingData.xls”, try the following:
  - Impute data into missing rows

## Checkpoint 10

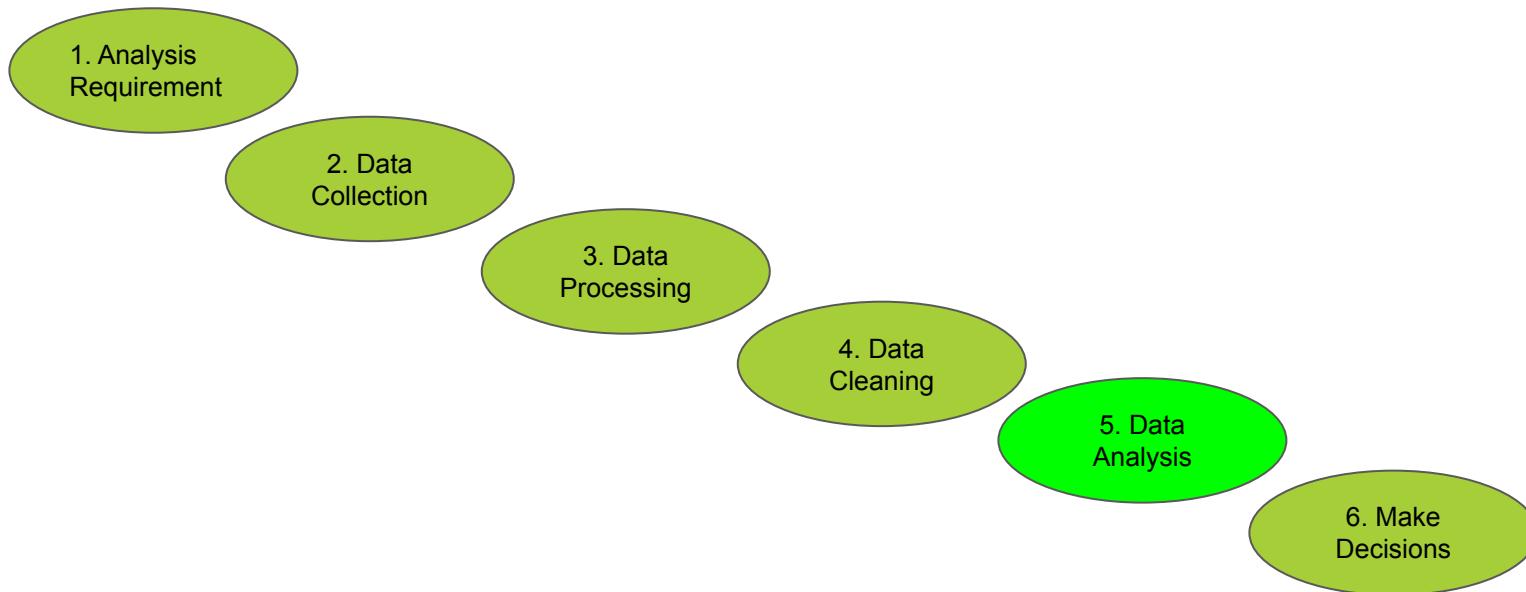
{<coding:lab>}



- Every student must be able to:
  - Impute missing data into a DataFrame
- For those who are waiting, try the following:
  - What are some other differences between removal and imputation?

# Data Analytics Process

{<coding:lab>}



# Data Analysis

{<coding:lab>}



- Two methods of Data Analysis
- Quantitative
  - This includes applying statistical methods to analyze data
- Graphical
  - This is all about visualising the data
- We will focus on graphical analysis in this session

# Data Visualisation

{<coding:lab>}



- Data visualization is the graphical representation of information and data.
  - Visual elements like charts, graphs, and maps are used
- Provides an accessible way to see and understand trends and patterns in data
- Analyze massive amounts of information and make data-driven decisions

# Data Visualisation In Python

{<coding:lab>}



- Many well-known libraries which provide data visualisation tools on the internet
  - Matplotlib
- Our main focus is going to be on Plotly

# What Is Plotly?

{<coding:lab>}



- A Python graphing library which enables users to create interactive graphs
- Allows user to share their data plotted with others

# Why Plotly?

{<coding:lab>}



- Equipped with a robust API
  - Users can create extremely fun and interactive plots
- Unlike other Python graphing libraries, Plotly has both online and offline mode
  - Online mode is a straightforward method for users to view their plots immediately
  - Offline mode sends the user a html link to be saved in a local file and viewed subsequently
  -

# Installing Plotly On Google Colab {<coding:lab>}



- Install plotly

```
1 !pip install plotly
```

- Import necessary functions and set your credentials
  - Note that setting credentials is only required for online mode

```
1 import plotly.express as px
```

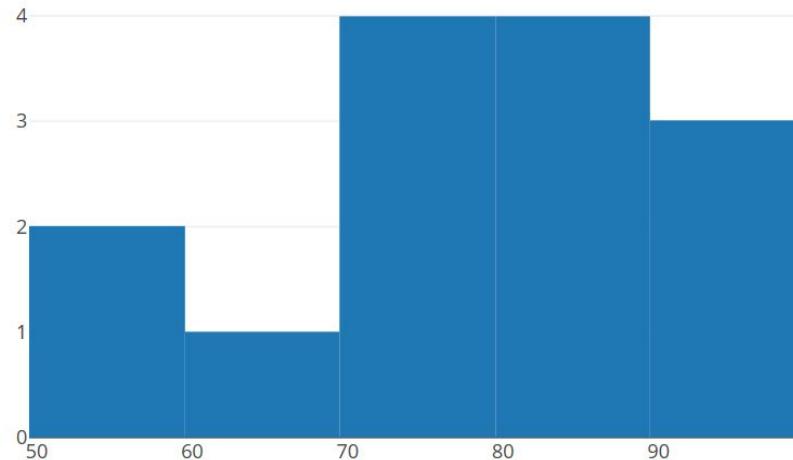
# Plotting Histogram (Demo/Practice{-<pp>ing:lab})



- Let's create a random list of scores and plot the graph on plotly by using `px.histogram()`

```
1 import pandas as pd
2 import plotly.express as px
3
4 x = [84, 65, 78, 75, 89, 59, 90, 88, 83, 72, 91, 90, 73, 54]
5 df = pd.DataFrame(x, columns =["x"])
6 #df
7 data = px.histogram(df,x="x")
8 data
```

{<coding:lab>}



## Checkpoint 11

{<coding:lab>}



- Every student must be able to:
  - Plot a Histogram
- For students who are waiting, try the following:
  - Explore Plotly
  - Try to plot another histogram with your own data
  - Beautify your graph!

# Plotting Boxplot (Demo/Practice - 12)oding:lab }

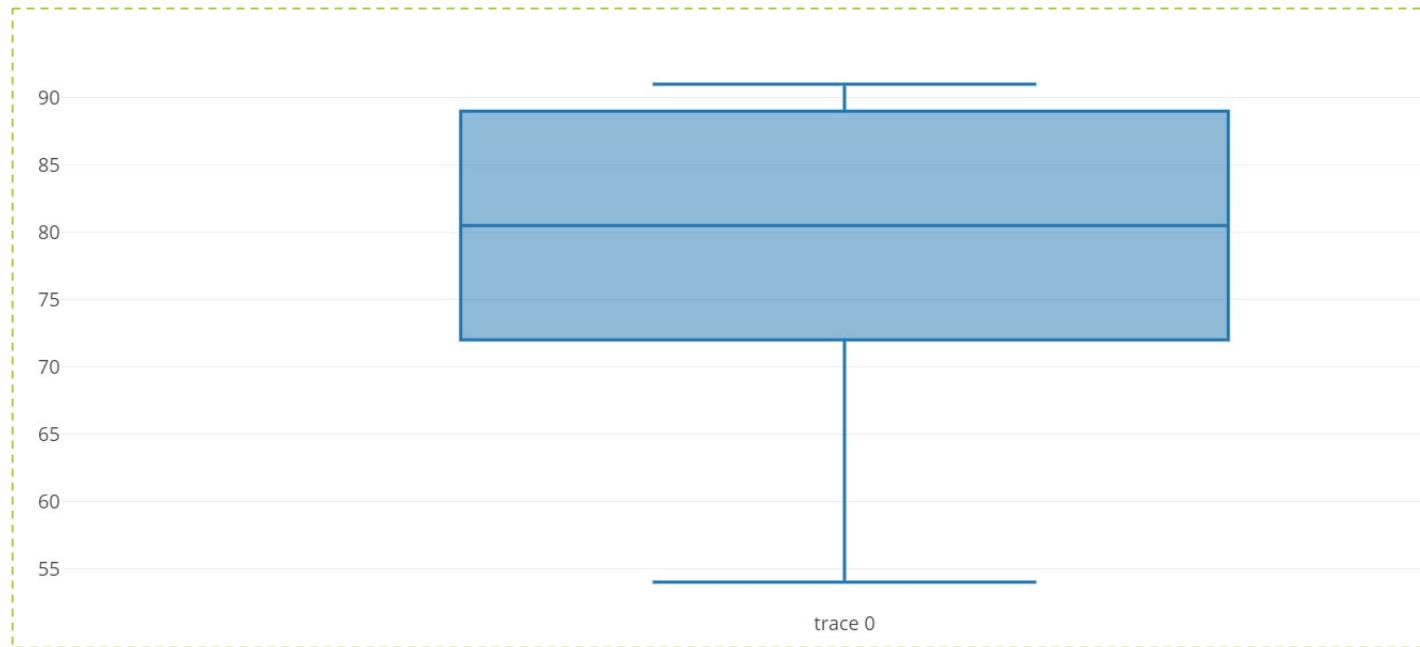


- Boxplot can be plotted when we use `px.box()`

```
1 import plotly.express as px
2 import pandas as pd
3 import numpy as np
4
5 y1 = [84, 65, 78, 75, 89, 59, 90, 88, 83, 72, 91, 90, 73, 54]
6 df = pd.DataFrame(y1,columns=['y1'])
7
8 boxplot = px.box(df,y='y1')
9 boxplot
```

- Try changing the y to x
- What if there are 2 sets of data? How do i plot 2 boxplots?

{<coding:lab>}



## Checkpoint 12

{<coding:lab>}



- Every student must be able to:
  - Plot a Boxplot
- For students who are waiting, try the following:
  - Explore Plotly
  - Try to plot another boxplot with your own data
  - Beautify your graph!

# Plotting Bar Graph (Demo/Practice {**13**)ing:lab }



- Bar Graph can be plotted with `px.bar()`

```
1 import plotly.express as px
2 import pandas as pd
3
4 foodItemList=['sushi', 'pizza', 'hamburger', 'fried chicken']
5 numberList=[2, 5, 4, 3]
6 df = pd.DataFrame({"Food Item":foodItemList, "Number of students":numberList})
7 data = px.bar(df,x="Food Item",y="Number of students")
8 data
```

{<coding:lab>}



## Checkpoint 13

{<coding:lab>}



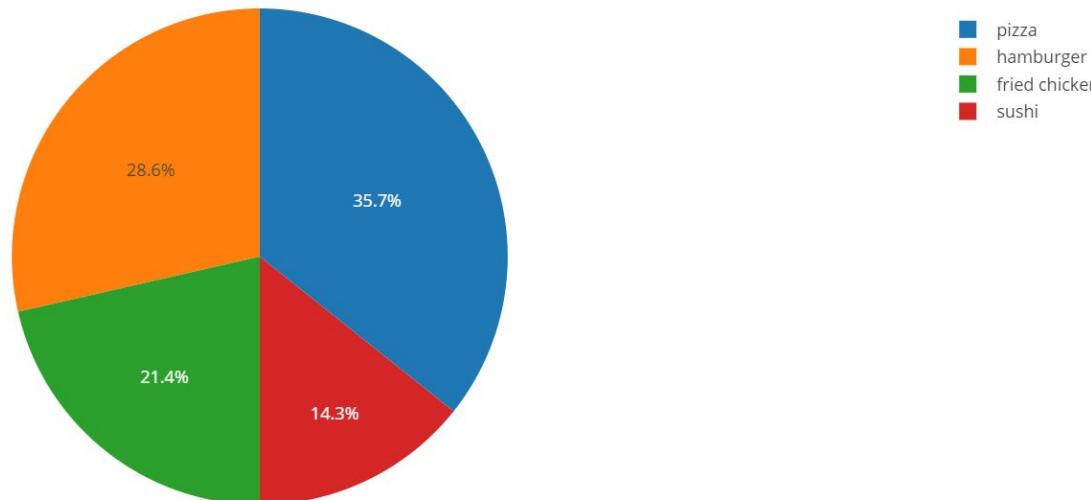
- Every student must be able to:
  - Plot a Bar Graph
  
- For students who are waiting, try the following:
  - Explore Plotly
  - Try to plot another bar graph with your own data
  - Beautify your graph!

# Plotting Pie Chart (Demo/Practice - 14)ding:lab }



- Pie Charts are plotted with `px.pie()` on plotly
- With the correct values typed in, the Pie Chart returns you the percentage taken up by each category

```
1 import plotly.express as px
2 import pandas as pd
3
4 foodItemList=['sushi', 'pizza', 'hamburger', 'fried chicken']
5 numberList=[2, 5, 4, 3]
6 df = pd.DataFrame({"Food Item":foodItemList, "Number of students":numberList})
7 #df
8
9 chart = px.pie(df,values="Number of students",names="Food Item")
10 chart
```



## Checkpoint 14

{<coding:lab>}



- Every student must be able to:
  - Plot a Pie Chart
  
- For students who are waiting, try the following:
  - Explore Plotly
  - Try to plot another pie chart with your own data
  - Beautify your graph!

# Summary

{<coding:lab>}



- Introduction to Pandas
  - Pandas DataFrame & Series
  - Extracting Row and Column from DataFrame
- Data Formats
  - Reading JSON, CSV, Excel and XML Data in Pandas

# Summary

{<coding:lab>}



- More Pandas Functions
  - Data preprocessing
  - Grouping and aggregation
  - Filtering
  - Joining data from multiple data frames
  - Pivoting
  - Outputting the data

# Summary

{<coding:lab>}



- Data Visualisation and Graphing
  - Visualisation and Graphing with Plotly
  - Histogram, Boxplot, Bar Graph, Pie Chart

{<coding:lab>}



# Appendix

---

# CSV Module in Python

{<coding:lab>}



- Python has a vast library of modules that are included with its distribution
  - csv module gives the Python programmer the ability to parse CSV files
  - The csv module will be able to read the vast majority of CSV files
  - Can also write CSV files using the csv module
- Documentation at: <https://docs.python.org/3/library/csv.html>

# JSON Format

{<coding:lab>}



- JSON (JavaScript Object Notation) is a lightweight data-interchange format
  - Easy for humans to read and write
  - Easy for machines to parse and generate

# JSON Library In Python

{<coding:lab>}



- The json library can parse JSON from strings or files
- The library parses JSON strings into a Python dictionary or list
  - Can also convert Python dictionaries or lists into JSON strings
- But since we already have Pandas, we can use the Pandas JSON reader instead

# Reading JSON in Pandas

{<coding:lab>}



- pd.read\_json() converts JSON strings to pandas object
- The following code snippet shows us how we can store a JSON string in a Dataframe:

```
1 import pandas as pd
2
3 json_str='{"country":"Netherlands","timezone":"Europe\Amsterdam","ip":"46.19.37.108",
4 |   "continent_code":"EU","longitude":5.75,"latitude":52.5,"country_code":"NL"}'
5
6 data = pd.read_json(json_str, typ='series')
7 print("Series\n", data)
```

# Importing JSON files

{<coding:lab>}



- We can also import a json file and print the content out into a Dataframe

```
1 import pandas as pd  
2 df = pd.read_json ('example_1.json',typ='series')  
3  
4 df
```

# JSON in Pandas (Demo/Practice - A1koding:lab }



- Create a JSON string which stores your personal information
  - Eg, First and last name, age, nationality
- Print the json string out into a Dataframe
  - Extract the first column of data from the Dataframe
    - It should be your first name

## Checkpoint A1

{<coding:lab>}



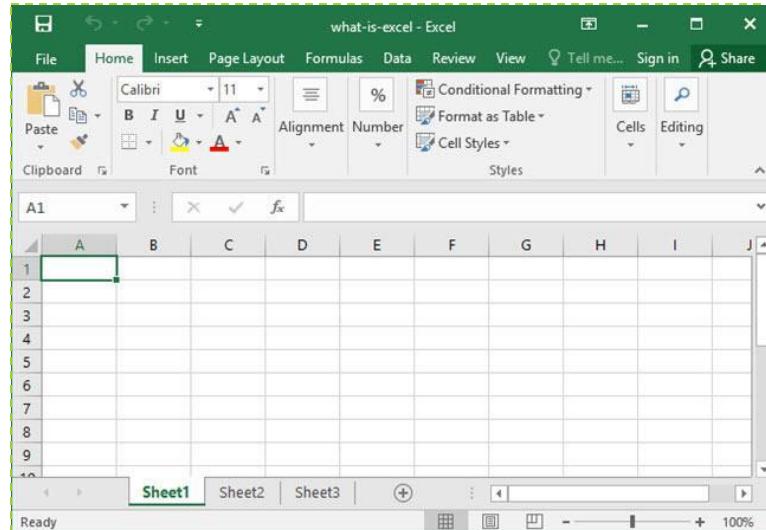
- Every student must be able to:
  - Convert a JSON string into a dataframe
  -
- For students who are waiting, try the following:
  - Create a dictionary and convert it to json format

# What is Excel?

{<coding:lab>}



- Spreadsheet developed by Microsoft
  - One of the most used software programs in the world



# Excel in Pandas

{<coding:lab>}



- Similar to all other data formats, we simply make use of built in method to access data from excel file
- Access the unit05\_data.xls file

```
1 import pandas as pd  
2 df = pd.read_excel ('unit05_data.xls',typ='series')  
3  
4 df
```

# Extracting data from xls file on Dataframe

{<coding:lab>}



- Regardless of whichever data format the file was originally in, once the data is stored in a Dataframe, the extraction process is the same
- Accessing the xls file, to extract the females born in the United States

```
1 import pandas as pd
2 df = pd.read_excel ('unit05_data.xls',typ='series')
3
4 gender_and_country = df[ (df['Gender'] == 'Female') & (df['Country'] == 'United States') ]
5 gender_and_country
```

# XLS file in Pandas (Demo/Practice - A2)

{<coding:lab>}



- Open up the XLS file 'Unit05-Demo53'
- The file contains the information of a group of people who visited a restaurant on a particular date
- The restaurant had a promotion on 21 May 2015, where customers 25 and above get to dine in for free
  - Find out the total number of people who enjoyed this perk

## Checkpoint A2

{<coding:lab>}



- Every student must be able to:
  - Extract data from xls file
  - Understand the use of bitwise operators
- For students who are waiting, try the following:
  - Pandas contains many useful methods which helps to compute statistics like df.mean(), df.median() and df.std()
  - Play around with these methods with all the previous statistical practices we have done earlier

# What Is XML?

{<coding:lab>}



- XML (eXtensible Markup Language) is another flexible format
  - Used to define and transfer data between computers
- Look at “Unit 05 - XML Sample Data.xml”
- XML files have a specific format

# Reading XML Files Using Python

{<coding:lab>}



- Using `xml.etree.ElementTree`
  - ET in short
- XML is a hierarchical data format
  - The most natural way to represent it is with a tree
- ET has two classes for this purpose
  - `ElementTree` represents the whole XML document as a tree
  - `Element` represents a single node in this tree

# How XML Files Are Read

{<coding:lab>}



- Interactions with the whole document (reading and writing to/from files) are usually done on the ElementTree level
- Interactions with a single XML element and its sub-elements are done on the Element level

# Importing XML File Into Python

{<coding:lab>}



- Using 'Unit 05 - XML Sample Data.xml'
- We can import this data by reading from a file

```
1 import xml.etree.ElementTree as ET  
2 tree = ET.parse('Unit 05 - XML Sample Data.xml')  
3 root = tree.getroot()
```

# Reading XML Files Using Python

{<coding:lab>}



- The root object has children nodes over which we can iterate:

```
1 import xml.etree.ElementTree as ET
2 tree = ET.parse('Unit 05 - XML Sample Data.xml')
3 root = tree.getroot()
4 for child in root:
5     print(child.tag, child.attrib)
```

# Nested Children

{<coding:lab>}



- Children are nested, and we can access specific child nodes by index
  - E.g. root[0][1].text

```
country {'name' : 'Liechtenstein'}
country {'name' : 'Singapore'}
country {'name' : 'Panama'}
'2008'
```

# How XML Files Are Read

{<coding:lab>}



- Interactions with the whole document (reading and writing to/from files) are usually done on the ElementTree level
- Interactions with a single XML element and its sub-elements are done on the Element level

# Importing XML File Into Python

{<coding:lab>}



- Using 'Unit 05 - XML Sample Data.xml'
- We can import this data by reading from a file

```
1 import xml.etree.ElementTree as ET  
2 tree = ET.parse('Unit 05 - XML Sample Data.xml')  
3 root = tree.getroot()
```

# Reading XML Files Using Python

{<coding:lab>}



- The root object has children nodes over which we can iterate:

```
1 import xml.etree.ElementTree as ET
2 tree = ET.parse('Unit 05 - XML Sample Data.xml')
3 root = tree.getroot()
4 for child in root:
5     print(child.tag, child.attrib)
```

# Nested Children

{<coding:lab>}



- Children are nested, and we can access specific child nodes by index
  - E.g. root[0][1].text

```
country {'name' : 'Liechtenstein'}
country {'name' : 'Singapore'}
country {'name' : 'Panama'}
'2008'
```

# XML in Python (Demo/Practice - A3) <coding:lab }



- Try reading 'Unit 05 - XML Sample Data.xml'
  - Print the child tags and attributes

## Checkpoint A3

{<coding:lab>}



- Every Student must be able to:
  - Extract XML data from Python
- For students who are waiting, try the following:
  - Not all elements of the XML input will end up as elements of the parsed tree
    - Currently, this module skips over any XML comments, processing instructions, and document type declarations in the input.
    - Read it up yourself!



- It is fairly easy to import CSV, JSON and Excel files in Pandas dataframe by simply making use of the built in methods
- However there is no built in method to import XML to Pandas
  - We have to make use of standard XML package and do some extra work to convert the data to Pandas Dataframes

# Parsing XML file

{<coding:lab>}



- Recall how we parsed an XML file by using parse method
- Subsequently, we obtain the data from each element and store it in a series

```
1 import pandas as pd
2 import xml.etree.ElementTree as ET
3 tree = ET.parse('Unit 05 - XML Sample Data.xml')
4
5 for node in tree.getroot():
6     name = node.attrib.get('name')
7     rank = node.find('rank').text
8     year = node.find('year').text
9     gdppc = node.find('gdppc').text
```

# Defining Dataframe

{<coding:lab>}



- To create a dataframe, we first define the column names for each column
- Then we append our series into our Dataframe

```
|1  dfcols = ["country", "rank", "year", "gdppc"]
|2  df_xml = pd.DataFrame(columns=dfcols)
|3  df_xml = df_xml.append( pd.Series([name, rank, year, gdppc],index=dfcols) , ignore_index =True)
|4
|5  print(df_xml)
```

# XML DataFrame (Demo/Practice - A4)

{<coding:lab>}



- Access the xml file - breakfast\_menu.xml
- The file displays the name of the food, price, description and the calories
- Parse the xml file and extract the data and store the data in a Dataframe

|   | food                        | price  | description                                       | calories |
|---|-----------------------------|--------|---|----------|
| 0 | Belgian Waffles             | \$5.95 | Two of our famous Belgian Waffles with plenty ... | 650      |
| 1 | Strawberry Belgian Waffles  | \$7.95 | Light Belgian waffles covered with strawberry...  | 900      |
| 2 | Berry-Berry Belgian Waffles | \$8.95 | Light Belgian waffles covered with an assortme... | 900      |
| 3 | French Toast                | \$4.50 | Thick slices made from our homemade sourdough ... | 600      |
| 4 | Homestyle Breakfast         | \$6.95 | Two eggs, bacon or sausage, toast, and our eve... | 950      |

## Checkpoint A4

{<coding:lab>}



- Every student must be able to:
  - Parse an XML file and extract data from each row
  - Append data into a dataframe
- For those who are waiting, try the following:
  - Read up on the functions available to import csv, json and excel files into Pandas Dataframe