**Data Engineering**

**Data engineering** is one of the most critical and foundational skills in any data scientist’s toolkit.

**Data Engineering Process**

There are several steps in Data Engineering process.

1. **Extract** - Data extraction is getting data from multiple sources. Ex. Data extraction from a website using Web scraping or gathering information from the data that are stored in different formats(JSON, CSV, XLSX etc.).
2. **Transform** - Tarnsforming the data means removing the data that we don't need for further analysis and converting the data in the format that all the data from the multiple sources is in the same format.
3. **Load** - Loading the data inside a data warehouse. Data warehouse essentially contains large volumes of data that are accessed to gather insights.

**Working with different file formats**

In the real-world, people rarely get neat tabular data. Thus, it is mandatory for any data scientist (or data engineer) to be aware of different file formats, common challenges in handling them and the best, most efficient ways to handle this data in real life. We have reviewed some of this content in other modules.

**File Format**

A file format is a standard way in which information is encoded for storage in a file. First, the file format specifies whether the file is a binary or ASCII file. Second, it shows how the information is organized. For example, the comma-separated values (CSV) file format stores tabular data in plain text.

To identify a file format, you can usually look at the file extension to get an idea. For example, a file saved with name “Data” in “CSV” format will appear as “Data.csv”. By noticing the “.csv” extension, we can clearly identify that it is a “CSV” file and the data is stored in a tabular format.

There are various formats for a dataset, .csv, .json, .xlsx etc. The dataset can be stored in different places, on your local machine or sometimes online.

**In this section, you will learn how to load a dataset into our Jupyter Notebook.**

Now, we will look at some file formats and how to read them in Python:

**Comma-separated values (CSV) file format**

The **Comma-separated values** file format falls under a spreadsheet file format.

In a spreadsheet file format, data is stored in cells. Each cell is organized in rows and columns. A column in the spreadsheet file can have different types. For example, a column can be of string type, a date type, or an integer type.

Each line in CSV file represents an observation, or commonly called a record. Each record may contain one or more fields which are separated by a comma.

**Reading data from CSV in Python**

The **Pandas** Library is a useful tool that enables us to read various datasets into a Pandas data frame

Let us look at how to read a CSV file in Pandas Library.

We use **pandas.read\_csv()** function to read the csv file. In the parentheses, we put the file path along with a quotation mark as an argument, so that pandas will read the file into a data frame from that address. The file path can be either a URL or your local file address.

**import** piplite

**await** piplite.install(['seaborn', 'lxml', 'openpyxl'])

​

**import** pandas **as** pd

**from** pyodide.http **import** pyfetch

​

filename **=** "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/addresses.csv"

​

**async** **def** download(url, filename):

response **=** **await** pyfetch(url)

**if** response.status **==** 200:

**with** open(filename, "wb") **as** f:

f.write(**await** response.bytes())

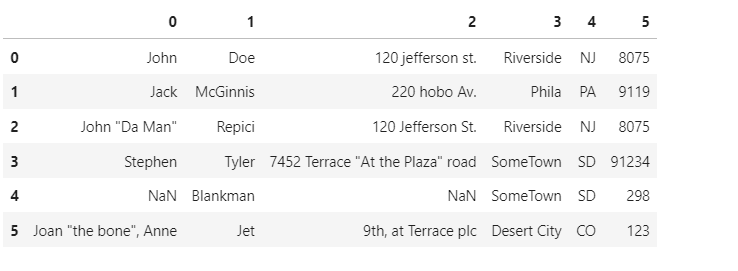
​

**await** download(filename, "addresses.csv")

​

df **=** pd.read\_csv("addresses.csv", header**=None**)

df

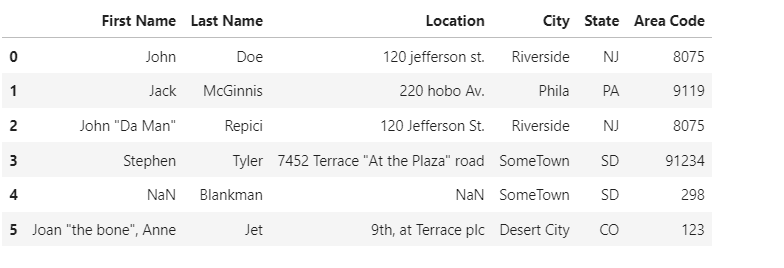


**Adding column name to the DataFrame**

We can add columns to an existing DataFrame using its **columns** attribute.

df.columns **=**['First Name', 'Last Name', 'Location ', 'City','State','Area Code']

df



**Selecting a single column**

To select the first column 'First Name', you can pass the column name as a string to the indexing operator.

df["First Name"]

0 John

1 Jack

2 John "Da Man"

3 Stephen

4 NaN

5 Joan "the bone", Anne

Name: First Name, dtype: object

**Selecting multiple columns**

To select multiple columns, you can pass a list of column names to the indexing operator.

df **=** df[['First Name', 'Last Name', 'Location ', 'City','State','Area Code']]

df

**Selecting rows using .iloc and .loc**

Now, let's see how to use .loc for selecting rows from our DataFrame.

**loc() : loc() is label based data selecting method which means that we have to pass the name of the row or column which we want to select.**

*# To select the first row*

df.loc[0]

First Name John

Last Name Doe

Location 120 jefferson st.

City Riverside

State NJ

Area Code 8075

Name: 0, dtype: object

[ ]:

*# To select the 0th,1st and 2nd row of "First Name" column only*

df.loc[[0,1,2], "First Name" ]

0 John

1 Jack

2 John "Da Man"

Name: First Name, dtype: object

Now, let's see how to use .iloc for selecting rows from our DataFrame.

**iloc() : iloc() is a indexed based selecting method which means that we have to pass integer index in the method to select specific row/column.**

*# To select the 0th,1st and 2nd row of "First Name" column only*

df.iloc[[0,1,2], 0]

0 John

1 Jack

2 John "Da Man"

Name: First Name, dtype: object

For more information please read the [documentation](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01).

Let perform some basic transformation in pandas.

**Transform Function in Pandas**

Python’s Transform function returns a self-produced dataframe with transformed values after applying the function specified in its parameter.

Let's see how Transform function works.

*#import library*

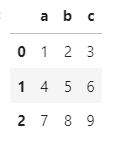
**import** pandas **as** pd

**import** numpy **as** np

*#creating a dataframe*

df**=**pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), columns**=**['a', 'b', 'c'])

df



Let’s say we want to add 10 to each element in a dataframe:

*#applying the transform function*

df **=** df.transform(func **=** **lambda** x : x **+** 10)

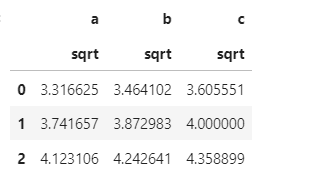
df



Now we will use DataFrame.transform() function to find the square root to each element of the dataframe.

result **=** df.transform(func **=** ['sqrt'])

result



For more information about the **transform()** function please read the [documentation](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.transform.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01).

**JSON file Format**

**JSON (JavaScript Object Notation)** is a lightweight data-interchange format. It is easy for humans to read and write.

JSON is built on two structures:

1. A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. It is a very common data format with a diverse range of applications.

The text in JSON is done through quoted string which contains the values in key-value mappings within { }. It is similar to the dictionary in Python.

Python supports JSON through a built-in package called **json**. To use this feature, we import the json package in Python script.

**import** json

**Writing JSON to a File**

This is usually called **serialization**. It is the process of converting an object into a special format which is suitable for transmitting over the network or storing in file or database.

To handle the data flow in a file, the JSON library in Python uses the **dump()** or **dumps()** function to convert the Python objects into their respective JSON object. This makes it easy to write data to files.

**import** json

person **=** {

'first\_name' : 'Mark',

'last\_name' : 'abc',

'age' : 27,

'address': {

"streetAddress": "21 2nd Street",

"city": "New York",

"state": "NY",

"postalCode": "10021-3100"

}

}

**serialization using dump() function**

**json.dump()** method can be used for writing to JSON file.

Syntax: json.dump(dict, file\_pointer)

Parameters:

1. **dictionary** – name of the dictionary which should be converted to JSON object.
2. **file pointer** – pointer of the file opened in write or append mode.

**with** open('person.json', 'w') **as** f: *# writing JSON object*

json.dump(person, f)

**serialization using dumps() function**

**json.dumps()** that helps in converting a dictionary to a JSON object.

It takes two parameters:

1. **dictionary** – name of the dictionary which should be converted to JSON object.
2. **indent** – defines the number of units for indentation

*# Serializing json*

json\_object **=** json.dumps(person, indent **=** 4)

*# Writing to sample.json*

**with** open("sample.json", "w") **as** outfile:

outfile.write(json\_object)

print(json\_object)

{

"first\_name": "Mark",

"last\_name": "abc",

"age": 27,

"address": {

"streetAddress": "21 2nd Street",

"city": "New York",

"state": "NY",

"postalCode": "10021-3100"

}

}

Our Python objects are now serialized to the file. To deserialize it back to the Python object, we use the load() function.

**Reading JSON to a File**

This process is usually called **Deserialization** - it is the reverse of serialization. It converts the special format returned by the serialization back into a usable object.

**Using json.load()**

The JSON package has json.load() function that loads the json content from a json file into a dictionary.

It takes one parameter:

File pointer: A file pointer that points to a JSON file.

**import** json

*# Opening JSON file*

**with** open('sample.json', 'r') **as** openfile:

*# Reading from json file*

json\_object **=** json.load(openfile)

print(json\_object)

print(type(json\_object))

{'first\_name': 'Mark', 'last\_name': 'abc', 'age': 27, 'address': {'streetAddress': '21 2nd Street', 'city': 'New York', 'state': 'NY', 'postalCode': '10021-3100'}}

<class 'dict'>

**XLSX file format**

**XLSX** is a Microsoft Excel Open XML file format. It is another type of Spreadsheet file format.

In XLSX data is organized under the cells and columns in a sheet.

**Reading the data from XLSX file**

Let’s load the data from XLSX file and define the sheet name. For loading the data you can use the Pandas library in python.

**import** pandas **as** pd

*# Not needed unless you're running locally*

*# import urllib.request*

*# urllib.request.urlretrieve("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/file\_example\_XLSX\_10.xlsx", "sample.xlsx")*

​

filename **=** "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/file\_example\_XLSX\_10.xlsx"

​

**async** **def** download(url, filename):

response **=** **await** pyfetch(url)

**if** response.status **==** 200:

**with** open(filename, "wb") **as** f:

f.write(**await** response.bytes())

​

**await** download(filename, "file\_example\_XLSX\_10.xlsx")

​

df **=** pd.read\_excel("file\_example\_XLSX\_10.xlsx")

df

**XML file format**

**XML is also known as Extensible Markup Language**. As the name suggests, it is a markup language. It has certain rules for encoding data. XML file format is a human-readable and machine-readable file format.

Pandas does not include any methods to read and write XML files. Here, we will take a look at how we can use other modules to read data from an XML file, and load it into a Pandas DataFrame.

**Writing with xml.etree.ElementTree**

The **xml.etree.ElementTree** module comes built-in with Python. It provides functionality for parsing and creating XML documents. ElementTree represents the XML document as a tree. We can move across the document using nodes which are elements and sub-elements of the XML file.

For more information please read the [xml.etree.ElementTree](https://docs.python.org/3/library/xml.etree.elementtree.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01" \t "_blank) documentation.

**import** xml.etree.ElementTree **as** ET

​

*# create the file structure*

employee **=** ET.Element('employee')

details **=** ET.SubElement(employee, 'details')

first **=** ET.SubElement(details, 'firstname')

second **=** ET.SubElement(details, 'lastname')

third **=** ET.SubElement(details, 'age')

first.text **=** 'Shiv'

second.text **=** 'Mishra'

third.text **=** '23'

​

*# create a new XML file with the results*

mydata1 **=** ET.ElementTree(employee)

*# myfile = open("items2.xml", "wb")*

*# myfile.write(mydata)*

**with** open("new\_sample.xml", "wb") **as** files:

mydata1.write(files)

**Reading with xml.etree.ElementTree**

Let's have a look at a one way to read XML data and put it in a Pandas DataFrame. You can see the XML file in the Notepad of your local machine.

*# Not needed unless running locally*

*# !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Sample-employee-XML-file.xml*

​

**import** xml.etree.ElementTree **as** etree

​

filename **=** "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Sample-employee-XML-file.xml"

​

**async** **def** download(url, filename):

response **=** **await** pyfetch(url)

**if** response.status **==** 200:

**with** open(filename, "wb") **as** f:

f.write(**await** response.bytes())

​

**await** download(filename, "Sample-employee-XML-file.xml")

You would need to firstly parse an XML file and create a list of columns for data frame, then extract useful information from the XML file and add to a pandas data frame.

Here is a sample code that you can use.:

tree **=** etree.parse("Sample-employee-XML-file.xml")

​

root **=** tree.getroot()

columns **=** ["firstname", "lastname", "title", "division", "building","room"]

​

datatframe **=** pd.DataFrame(columns **=** columns)

​

**for** node **in** root:

​

firstname **=** node.find("firstname").text

​

lastname **=** node.find("lastname").text

​

title **=** node.find("title").text

division **=** node.find("division").text

building **=** node.find("building").text

room **=** node.find("room").text

datatframe **=** datatframe.append(pd.Series([firstname, lastname, title, division, building, room], index **=** columns), ignore\_index **=** **True**)

datatframe

**Reading xml file using pandas.read\_xml function**

We can also read the downloaded xml file using the read\_xml function present in the pandas library which returns a Dataframe object.

For more information read the [pandas.read\_xml](https://pandas.pydata.org/pandas-docs/dev/reference/api/pandas.read_xml.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01" \l "pandas-read-xml" \t "_blank) documentation.

*# Herein xpath we mention the set of xml nodes to be considered for migrating to the dataframe which in this case is details node under employees.*

df**=**pd.read\_xml("Sample-employee-XML-file.xml", xpath**=**"/employees/details")

**Save Data**

Correspondingly, Pandas enables us to save the dataset to csv by using the **dataframe.to\_csv()** method, you can add the file path and name along with quotation marks in the parentheses.

For example, if you would save the dataframe df as **employee.csv** to your local machine, you may use the syntax below:

datatframe.to\_csv("employee.csv", index**=False**)

We can also read and save other file formats, we can use similar functions to **pd.read\_csv()** and **df.to\_csv()** for other data formats. The functions are listed in the following table:

**Read/Save Other Data Formats**

| **Data Formate** | **Read** | **Save** |
| --- | --- | --- |
| csv | pd.read\_csv() | df.to\_csv() |
| json | pd.read\_json() | df.to\_json() |
| excel | pd.read\_excel() | df.to\_excel() |
| hdf | pd.read\_hdf() | df.to\_hdf() |
| sql | pd.read\_sql() | df.to\_sql() |
| ... | ... | ... |

Let's move ahead and perform some **Data Analysis**.

**Binary File Format**

"Binary" files are any files where the format isn't made up of readable characters. It contain formatting information that only certain applications or processors can understand. While humans can read text files, binary files must be run on the appropriate software or processor before humans can read them.

Binary files can range from image files like JPEGs or GIFs, audio files like MP3s or binary document formats like Word or PDF.

Let's see how to read an **Image** file.

**Reading the Image file**

Python supports very powerful tools when it comes to image processing. Let’s see how to process the images using the **PIL** library.

PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities.

*# importing PIL*

**from** PIL **import** Image

​

*# Uncomment if running locally*

*# import urllib.request*

*# urllib.request.urlretrieve("https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg", "dog.jpg")*

​

filename **=** "https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg"

​

**async** **def** download(url, filename):

response **=** **await** pyfetch(url)

**if** response.status **==** 200:

**with** open(filename, "wb") **as** f:

f.write(**await** response.bytes())

​

**await** download(filename, "dog.jpg")

*# Read image*

img **=** Image.open('dog.jpg')

*# Output Images*

display(img)

**Data Analysis**

In this section, you will learn how to approach data acquisition in various ways and obtain necessary insights from a dataset. By the end of this lab, you will successfully load the data into Jupyter Notebook and gain some fundamental insights via the Pandas Library.

In our case, the **Diabetes Dataset** is an online source and it is in CSV (comma separated value) format. Let's use this dataset as an example to practice data reading.

**About this Dataset**

**Context:** This dataset is originally from the **National Institute of Diabetes and Digestive and Kidney Diseases**. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years of age of Pima Indian heritage.

**Content:** The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

We have 768 rows and 9 columns. The first 8 columns represent the features and the last column represent the target/label.

*# Import pandas library*

**import** pandas **as** pd

filename **=** "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/diabetes.csv"

​

**async** **def** download(url, filename):

response **=** **await** pyfetch(url)

**if** response.status **==** 200:

**with** open(filename, "wb") **as** f:

f.write(**await** response.bytes())

​

**await** download(filename, "diabetes.csv")

df **=** pd.read\_csv("diabetes.csv")

After reading the dataset, we can use the **dataframe.head(n)** method to check the top n rows of the dataframe, where n is an integer. Contrary to **dataframe.head(n)**, **dataframe.tail(n)** will show you the bottom n rows of the dataframe.

*# show the first 5 rows using dataframe.head() method*

print("The first 5 rows of the dataframe")

df.head(5)

To view the dimensions of the dataframe, we use the **.shape** parameter.

df.shape

**Statistical Overview of dataset**

df.info()

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

df.describe()

Pandas **describe()** is used to view some basic statistical details like percentile, mean, standard deviation, etc. of a data frame or a series of numeric values. When this method is applied to a series of strings, it returns a different output

**Identify and handle missing values**

We use Python's built-in functions to identify these missing values. There are two methods to detect missing data:

**.isnull()**

**.notnull()**

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

missing\_data **=** df.isnull()

missing\_data.head(5)

"True" stands for missing value, while "False" stands for not missing value.

**Count missing values in each column**

Using a for loop in Python, we can quickly figure out the number of missing values in each column. As mentioned above, "True" represents a missing value, "False" means the value is present in the dataset. In the body of the for loop the method ".value\_counts()" counts the number of "True" values.

**for** column **in** missing\_data.columns.values.tolist():

print(column)

print (missing\_data[column].value\_counts())

print("")

As you can see above, there is no missing values in the dataset.

**Correct data format**

Check all data is in the correct format (int, float, text or other).

In Pandas, we use

**.dtype()** to check the data type

**.astype()** to change the data type

Numerical variables should have type **'float'** or **'int'**.

df.dtypes

As we can see above, All columns have the correct data type.

**Visualization**

**Visualization** is one of the best way to get insights from the dataset. **Seaborn** and **Matplotlib** are two of Python's most powerful visualization libraries.

*# import libraries*

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

labels**=** 'Diabetic','Not Diabetic'

plt.pie(df['Outcome'].value\_counts(),labels**=**labels,autopct**=**'%0.02f%%')

plt.legend()

plt.show()

As you can see above, 65.10% females are Diabetic and 34.90% are Not Diabetic.