CSV Files :

Pros :

- Easy to export/import from Excel.

Cons :

- No type info, everything is string.
- No standard.
- Hard to work with unicode.

Creating our custom csv reader module :

```
def iter_records(file_name):
    with bz2.open(file_name. ‘rt’) as fp:
        reader = csv.DictReader(fp)
        for csv_record in reader:
            record = {}
            for col in columns:
                value = csv_record[col.src]
                record[col.dest] = col.convert(value)
            yield record

def custom_csv_reader():
    from pprint import pprint

    for i, record in enumerate(iter_records(‘filename.csv.bz2’)):
        if i>= 10 :
            break
        pprint(record)
```

Using Pandas inbuilt csv reader:

//assign a series with all columns having dates into the parse_dates argument in read_csv function

```
time_cols = [‘column_with_dates1’, …….]
df = pd.read_csv(‘filename.csv.bz2’, parse_dates=time_cols)
```

XML Files:

DOM – Document Object Model

Everything into memory

SAX – Simple API for XML

Iterative (good for big files)

Parsers:

- ElementTree – in the standard library
- lxml – third party

Using ElementTree to make a load_XML module :

```
def iter_rides(file_name):
    with bz2.open(file_name, 'rt') as fp:
        tree = xml.parse(fp)
    rides = tree.getroot()
    for elem in rides:
        record = {}
        for tag, func in conversion:
            text = elem.find(tag).text
            record[tag] = func(text)
        yield record

def load_xml(file_name):
    records = iter_rides(file_name)
    return pd.Dataframe.from_records(records)

//Example
if __name__ == '__main__':
    df = load_xml('taxi.xml.bz2')
    print(df.head())
```

Parquet, Avro and ORC:

- Big data systems sometimes store data in files.
- Text is a bottleneck (not efficient)
- Better formats: Parquet, Avro, ORC
- Python libraries for reading all of these formats.

//Example(Reading our taxi.parquet file)

```
import pyarrow.parquet as pq
table = pq.read_table('taxi.parquet')
df = table.to_pandas()
df.dtypes
```

Working with Unstructured text:

You have to learn Regex(Regular expression) from our tutorials for dealing with Unstructured text.
For searching for logs with regex you can use the **pythex** library.

JSON files :

- JSON = Javascript Object Notation
- Supported in many languages.
- Not all Python types can be encoded in JSON.
- One of the most common serialization formats in APIs.

Making HTTP calls:

- RPC = Remote Procedure Call
- JSON over HTTP is very common.
- Python has a built in URL open function in urllib.request (You would need some low level JSON parsing skills for this).

Calling HTTP + JSON API using requests:

```
import requests
url = ‘http://localhost:8989/trips’
query = {
        ‘start’ : ‘2018-11-01T00:02:04’,
        ‘end’ : ‘2018-11-01T00:44:51’,
}
headers = {
        ‘x-trips-token’: ‘l3tm3in’,
}
resp = requests.get(url, params=query, headers=headers)
if not resp.ok:
        raise SystemExit(f’FAIL: {resp.status_code} – {resp.reson}’)

reply = resp.json()
if not reply[‘ok’] or ‘trips’ not in reply:
        raise SystemExit(f’bad reply – {reply}’)
count = len(reply[‘trips’])
print(f’Total of {count} trips found’)
```

Best tools for Web Scraping : (Learn these in detail in our courses)

- Beautiful Soup
- Selenium
- Scrapy

What should be in Schema / Metadata ?

- Description
- Types
- Units
- Constraints
- Inter field constraints
- Relations

Types of Databases :

- **Relational:** PostgreSQL, MySQL, MSSQL, Oracle
- **Key/Value:** Redis, Memcached
- **Document:** Elasticsearch, MongoDB
- **Graph:** Neo4j, Dgraph