Parsing Company 10Ks From the SEC

In this module, now that we can grab any filing we want from the daily-index filings we are going to move on to the next topic parsing financial documents. The easiest one we can start with is the 10K because the underlying structure provided to us will make grabbing the data accessible and quick. We will only focus on the data tables as this is separated from the document itself. However, in time, we will explore how to parse the different components of the 10K.

Import the libraries

This module will require only three libraries, the first is the `requests` library for making the URL requests, `bs4` to parse the files and content, and finally `pandas` which will be used for taking our cleaned data and giving it structure.

In [1]:
```
# import our libraries
import requests
import pandas as pd
from bs4 import BeautifulSoup
```

Grab the Filing XML Summary

Something that makes 10-K and for that matter 10-Q filings so unique is we have access to a particular document that gives us a quick way to grab the data we need from a 10-K. This file is the **filing summary** and comes in an either an `XML` or `xlsx` format. While you would think these two files would be identical, they are not, the `XML` version of the file provides us with a quick way to see the structure of the 10-K, defines whether a section is a note, table or details, and the name and each corresponding file for each section.

The `xlsx` file, on the other hand, contains each section of the 10K in an excel style format. This file can come in handy if we want to parse just a single location, but be warned that formatting issues will not make it a simple load.

Let's assume we want to parse the `XML` file as we want to leverage the underlying structure of the 10-K report. In the section below, I outline how you would go about this process and use a sample document URL for our demonstration.

In [9]:
```
# define the base url needed to create the file url.
base_url = r"https://www.sec.gov"

# convert a normal url to a document url
normal_url = r"https://www.sec.gov/Archives/edgar/data/1265107/0001265107-19-0
00004.txt"
normal_url = normal_url.replace('-','').replace('.txt','/index.json')

# define a url that leads to a 10k document landing page
documents_url = r"https://www.sec.gov/Archives/edgar/data/1265107/000126510719
000004/index.json"

# request the url and decode it.
content = requests.get(documents_url).json()

for file in content['directory']['item']:

    # Grab the filing summary and create a new url leading to the file so we c
an download it.
    if file['name'] == 'FilingSummary.xml':

        xml_summary = base_url + content['directory']['name'] + "/" + file['na
me']

        print('-' * 100)
        print('File Name: ' + file['name'])
        print('File Path: ' + xml_summary)
```

```
----------------------------------------------------------------------------
----------------------
File Name: FilingSummary.xml
File Path: https://www.sec.gov/Archives/edgar/data/1265107/00012651071900000
4/FilingSummary.xml
```

Parsing the Filing Summary

Okay, we now have access to a filing summary file. The first thing we need to do is request the file using the `requests` library we will then take the contents of that request and pass through our `BeautifulSoup` object. I encourage individuals who are new to this process to look at the file itself, so you better understanding of the structure, this will reinforce my approach below.

The main section of the file we want to grab belongs under the `myreports` tag. This contains a list of each of the reports in the document. Each report falls under a `report` tag and has the following structure:

```
<Report instance="mtii-20181231.xml">
    <IsDefault>false</IsDefault>
    <HasEmbeddedReports>false</HasEmbeddedReports>
    <HtmlFileName>R1.htm</HtmlFileName>
    <LongName>0001000 - Document - Document and Entity Information</LongName>
    <ReportType>Sheet</ReportType>
    <Role>http://www.monitronics.com/role/DocumentAndEntityInformation</Role>
    <ShortName>Document and Entity Information</ShortName>
    <MenuCategory>Cover</MenuCategory>
    <Position>1</Position>
</Report>
```

The main tags we will be concerned with are the following:

1. **`HtmlFileName` -** This is the name of the file and will be needed to build the file URL.
2. **`LongName` -** This is the long name of the report, with its ID. Keep in mind the ID can be leveraged in other 10Ks of other companies, but unfortunately, it is not always guaranteed.
3. **`ShortName` -** The short name of the report, this is surprisingly more consistent across companies compared to the long name which includes the ID.
4. **`MenuCategory` -** This can be thought of as a category the report falls under, a table, notes, details, cover, or statements. This will be leveraged as another filtering mechanism.
5. **`Position` -** This is the position of the report in the main document and also corresponds to the `HtmlFileName.`

In [3]:
```
# define a new base url that represents the filing folder. This will come in h
andy when we need to download the reports.
base_url = xml_summary.replace('FilingSummary.xml', '')

# request and parse the content
content = requests.get(xml_summary).content
soup = BeautifulSoup(content, 'lxml')

# find the 'myreports' tag because this contains all the individual reports su
bmitted.
reports = soup.find('myreports')

# I want a list to store all the individual components of the report, so creat
e the master list.
master_reports = []

# loop through each report in the 'myreports' tag but avoid the last one as th
is will cause an error.
for report in reports.find_all('report')[:-1]:

    # let's create a dictionary to store all the different parts we need.
    report_dict = {}
    report_dict['name_short'] = report.shortname.text
    report_dict['name_long'] = report.longname.text
    report_dict['position'] = report.position.text
    report_dict['category'] = report.menucategory.text
    report_dict['url'] = base_url + report.htmlfilename.text

    # append the dictionary to the master list.
    master_reports.append(report_dict)

    # print the info to the user.
    print('-'*100)
    print(base_url + report.htmlfilename.text)
    print(report.longname.text)
    print(report.shortname.text)
    print(report.menucategory.text)
    print(report.position.text)
```

```
Income Taxes - Schedule of Unrecognized Tax Benefit Roll-Forward (Details)
Details
69
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R70.htm
2413402 - Disclosure - Stock-based and Long-Term Compensation - Narrative (De
tails)
Stock-based and Long-Term Compensation - Narrative (Details)
Details
70
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R71.htm
2413403 - Disclosure - Stock-based and Long-Term Compensation - Schedule of O
ption Activity (Details)
Stock-based and Long-Term Compensation - Schedule of Option Activity (Detail
s)
Details
71
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R72.htm
2413404 - Disclosure - Stock-based and Long-Term Compensation - Schedule of U
nvested Restricted Stock Awards and Restricted Stock Units (Details)
Stock-based and Long-Term Compensation - Schedule of Unvested Restricted Stoc
k Awards and Restricted Stock Units (Details)
Details
72
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R73.htm
2414402 - Disclosure - Stockholder's Equity - Narrative (Details)
Stockholder's Equity - Narrative (Details)
Details
73
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R74.htm
2414403 - Disclosure - Stockholder's Equity - Roll Forward Activity (Details)
Stockholder's Equity - Roll Forward Activity (Details)
Details
74
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R75.htm
2415401 - Disclosure - Employee Benefit Plans - Narrative (Details)
Employee Benefit Plans - Narrative (Details)
Details
75
-----------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R76.htm
2416402 - Disclosure - Commitments, Contingencies and Other Liabilities - Sum
mary of Future Minimum Lease Payments (Details)
Commitments, Contingencies and Other Liabilities - Summary of Future Minimum
Lease Payments (Details)
```

```
Details
76
------------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R77.htm
2416403 - Disclosure - Commitments, Contingencies and Other Liabilities - Nar
rative (Details)
Commitments, Contingencies and Other Liabilities - Narrative (Details)
Details
77
------------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R78.htm
2417402 - Disclosure - Quarterly Financial Information (Unaudited - see accom
panying accountants' report) (Details)
Quarterly Financial Information (Unaudited - see accompanying accountants' re
port) (Details)
Details
78
------------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R79.htm
2418402 - Disclosure - Condensed Consolidating Balance Sheet (Details)
Condensed Consolidating Balance Sheet (Details)
Details
79
------------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R80.htm
2418403 - Disclosure - Condensed Consolidating Statement of Operations and Co
mprehensive Income (Loss) (Details)
Condensed Consolidating Statement of Operations and Comprehensive Income (Los
s) (Details)
Details
80
------------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R81.htm
2418404 - Disclosure - Condensed Consolidating Statements of Cash Flows (Deta
ils)
Condensed Consolidating Statements of Cash Flows (Details)
Details
81
------------------------------------------------------------------------------
----------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R9999.htm
Uncategorized Items - mtii-20181231.xml
Uncategorized Items - mtii-20181231.xml
Cover
82
```

Grabbing the Financial Statements

We now have a nice organized list of all the different components of the 10-K filing, while it won't have all the info it makes the process of getting the data tables a lot easier. We can always revisit the actual text but at this point let's move forward assuming that we want to get the company financial statements. This will include the following:

1. Balance Sheet
2. Statement of Cash Flows
3. Income Statement
4. Statement of Stock Holder Equity

The first thing we need to do is a loop through each report dictionary, see if the financial statement we are looking for exists in that dictionary and if it does append it to a new list called `url_list`. The `url_list` will contain a URL to each of the statements, and each statement will exist in an HTML format that we can scrape relatively quickly.

As a side note, I will be working on a list of report naming conventions across companies. This way, for example, if we want to find the balance sheet we have a list of potential names and IDs we can try.

In [4]:
```
# create the list to hold the statement urls
statements_url = []

for report_dict in master_reports:

    # define the statements we want to look for.
    item1 = r"Consolidated Balance Sheets"
    item2 = r"Consolidated Statements of Operations and Comprehensive Income
 (Loss)"
    item3 = r"Consolidated Statements of Cash Flows"
    item4 = r"Consolidated Statements of Stockholder's (Deficit) Equity"

    # store them in a list.
    report_list = [item1, item2, item3, item4]

    # if the short name can be found in the report list.
    if report_dict['name_short'] in report_list:

        # print some info and store it in the statements url.
        print('-'*100)
        print(report_dict['name_short'])
        print(report_dict['url'])

        statements_url.append(report_dict['url'])
```

```
----------------------------------------------------------------------------
----------------------
Consolidated Balance Sheets
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R2.htm
----------------------------------------------------------------------------
----------------------
Consolidated Statements of Operations and Comprehensive Income (Loss)
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R4.htm
----------------------------------------------------------------------------
----------------------
Consolidated Statements of Cash Flows
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R5.htm
----------------------------------------------------------------------------
----------------------
Consolidated Statements of Stockholder's (Deficit) Equity
https://www.sec.gov/Archives/edgar/data/1265107/000126510719000004/R6.htm
```

Scraping the Financial Statements

We now have each financial statement's URL that we can now request for the content of that specific statement. The first thing we will need to do is a loop through all the URLs, request each one, and then parse the content. Like the **filing xml summary** up above, I encourage individuals new to scraping the documents to visit each HTML file up above to see what the data looks like.

You'll first notice, it's a simple HTML table. Depending on the statement you're looking at, the structure may be slightly different, but a general hierarchy does exist. My approach to parsing the table falls into three significant steps:

1. Parsing the table headers.
2. Parsing the table rows.
3. Parsing the table sections.

I find this approach to be the most reliable as it allows us to loop through each row in the table, but we ask a specific question for each row. The question is, what type of row are you?

Depending on the answer to this question, we can determine which section of the `statement dictionary` the row should be inserted to. Table headers will contain important information regarding the time horizon of the financial statement. Table sections, help us to distinguish different parts of the statement easily.

Finally, table rows contain the data we want to parse. We distinguish these rows, by seeing if certain elements exist in each. For example, only header row would include a `th` tag inside of it; otherwise, it's not considered a header. Section headers contain a `strong` element but no `th` tags.

Hopefully, you're catching on to my approach when it comes to grabbing each row. I ask a simple question to distinguish each row. Once we have the row, we loop through each of the `td` tags, strip text, store it in a list using a list comprehension, and store it in the appropriate section of our statement dictionary.

In [5]:
```
# let's assume we want all the statements in a single data set.
statements_data = []

# loop through each statement url
for statement in statements_url:

    # define a dictionary that will store the different parts of the statemen
t.
    statement_data = {}
    statement_data['headers'] = []
    statement_data['sections'] = []
    statement_data['data'] = []

    # request the statement file content
    content = requests.get(statement).content
    report_soup = BeautifulSoup(content, 'html')

    # find all the rows, figure out what type of row it is, parse the element
s, and store in the statement file list.
    for index, row in enumerate(report_soup.table.find_all('tr')):

        # first let's get all the elements.
        cols = row.find_all('td')

        # if it's a regular row and not a section or a table header
        if (len(row.find_all('th')) == 0 and len(row.find_all('strong')) == 0
):
            reg_row = [ele.text.strip() for ele in cols]
            statement_data['data'].append(reg_row)

        # if it's a regular row and a section but not a table header
        elif (len(row.find_all('th')) == 0 and len(row.find_all('strong')) !=
0):
            sec_row = cols[0].text.strip()
            statement_data['sections'].append(sec_row)

        # finally if it's not any of those it must be a header
        elif (len(row.find_all('th')) != 0):
            hed_row = [ele.text.strip() for ele in row.find_all('th')]
            statement_data['headers'].append(hed_row)

        else:
            print('We encountered an error.')

    # append it to the master list.
    statements_data.append(statement_data)
```

Converting the Data into a Data Frame

Great, we now have all the data for all the financial statements, and it's in a much better structure that will allow us to work with it. We still have some work to do regarding transforming it into the right data type, but we will handle that later. Let's first get it into a data frame, and from there we need to do some massaging to the data.

The first thing we will notice is the index won't work with what we have so we take the first column which contains the indexes and set it as the index. Let's also make sure to rename the index column to something more meaningful.

From here, we need to remove certain characters before we do our type conversion. We can use the `replace` method specifying the `regex` parameter to true. I have to do three separate replace because one handles positive data, one handles negative data, and one handles blank values. After the regex, we can do a type conversion to the whole data frame and then assign our column headers.

In [6]:
```
# Grab the proper components
income_header =  statements_data[1]['headers'][1]
income_data = statements_data[1]['data']

# Put the data in a DataFrame
income_df = pd.DataFrame(income_data)

# Display
print('-'*100)
print('Before Reindexing')
print('-'*100)
display(income_df.head())

# Define the Index column, rename it, and we need to make sure to drop the old
column once we reindex.
income_df.index = income_df[0]
income_df.index.name = 'Category'
income_df = income_df.drop(0, axis = 1)

# Display
print('-'*100)
print('Before Regex')
print('-'*100)
display(income_df.head())

# Get rid of the '$', '(', ')', and convert the '' to NaNs.
income_df = income_df.replace('[\$,)]','', regex=True )\
                     .replace( '[(]','-', regex=True)\
                     .replace( '', 'NaN', regex=True)

# Display
print('-'*100)
print('Before type conversion')
print('-'*100)
display(income_df.head())

# everything is a string, so let's convert all the data to a float.
income_df = income_df.astype(float)

# Change the column headers
income_df.columns = income_header

# Display
print('-'*100)
print('Final Product')
print('-'*100)

# show the df
income_df

# drop the data in a CSV file if need be.
# income_df.to_csv('income_state.csv')
```