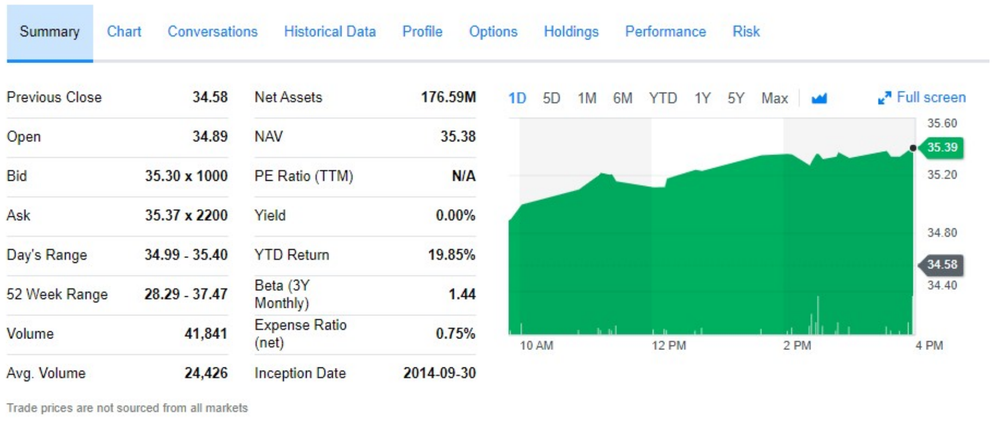
Web scraping Yahoo Finance

Yahoo Finance provides a tremendous amount of financial data related to a wide variety of financial instruments, in this tutorial, we will cover how to web scrape Yahoo Finance for ETF & mutual fund data. This tutorial will require two libraries, `requests` for requesting the URL, and `bs4` for parsing the HTML content of our request.

We will define our `base_url` to be **https://finance.yahoo.com (https://finance.yahoo.com)**, which is the main page. To query a particular instrument, we pass through the ticker symbol of our instrument. For example, if I want to query the `QQQ` ETF, then we would construct the following URL **https://finance.yahoo.com/quote/QQQ (https://finance.yahoo.com/quote/QQQ)**. Once, we construct our URL, we will request the content, pass the material through to our Beautiful Soup object, and then begin parsing the content.

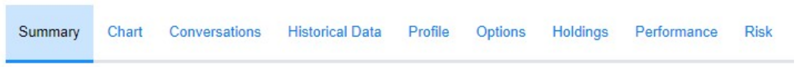
The main landing page for any ETF or Mutual fund contains items of interest for our analysis, the first being all the links to the additional data and a summary table describing our instrument. First, let's grab both the left and right side of that summary table. The summary appears as the following:



Previous Close	34.58	Net Assets	176.59M
Open	34.89	NAV	35.38
Bid	35.30 x 1000	PE Ratio (TTM)	N/A
Ask	35.37 x 2200	Yield	0.00%
Day's Range	34.99 - 35.40	YTD Return	19.85%
52 Week Range	28.29 - 37.47	Beta (3Y Monthly)	1.44
Volume	41,841	Expense Ratio (net)	0.75%
Avg. Volume	24,426	Inception Date	2014-09-30

Trade prices are not sourced from all markets

With the summary table captured, let's move on to the nav menu, this will contain all the other links to the different data sources. The nav menu appears as the following:



From here, loop through all the `a` tags, grab the corresponding `href` attribute and store it in a dictionary we create above the loop.

In [2]:
```python
import requests
from bs4 import BeautifulSoup
import numpy as np

# define a symbol to search
stock_symbol = "ARKQ"

# base url
base = "https://finance.yahoo.com"

# url to particular stock
endpoint = base + "/quote/{}".format(stock_symbol)

# request it & parse it
response = requests.get(endpoint).content
soup = BeautifulSoup(response, 'html.parser')

# these two sections contain the information for the summary page, so grab them to be parsed later.
left_summary_table = soup.find_all('div', {'data-test':'left-summary-table'})
right_summary_table = soup.find_all('div', {'data-test':'right-summary-table'})

# find the nav menu to get the other page links
nav_menu = soup.find('div', {'id':'quote-nav'})

# store the links in a dictionary
link_dictionary = {}

for anchor in nav_menu.find_all('a'):

    # grab the text (Page Name), link to page, and store the full link in the dictionary.
    text = anchor.text
    full_link = base + anchor['href']
    link_dictionary[text] = full_link


link_dictionary
```

```
Out[2]: {'Summary': 'https://finance.yahoo.com/quote/ARKQ?p=ARKQ',
         'Historical Data': 'https://finance.yahoo.com/quote/ARKQ/history?p=ARKQ',
         'Profile': 'https://finance.yahoo.com/quote/ARKQ/profile?p=ARKQ',
         'Options': 'https://finance.yahoo.com/quote/ARKQ/options?p=ARKQ',
         'Holdings': 'https://finance.yahoo.com/quote/ARKQ/holdings?p=ARKQ',
         'Performance': 'https://finance.yahoo.com/quote/ARKQ/performance?p=ARKQ',
         'Risk': 'https://finance.yahoo.com/quote/ARKQ/risk?p=ARKQ'}
```

If we split our list as we collect them, it will make storing the data more consistent. Let's define a function that takes two parameters, a list and the number of items we want in our chunk.

```
In [3]: # build our split list function
        def split_list(my_list, chunks):
            return [my_list[i:i + chunks] for i in range(0, len(my_list), chunks)]
```

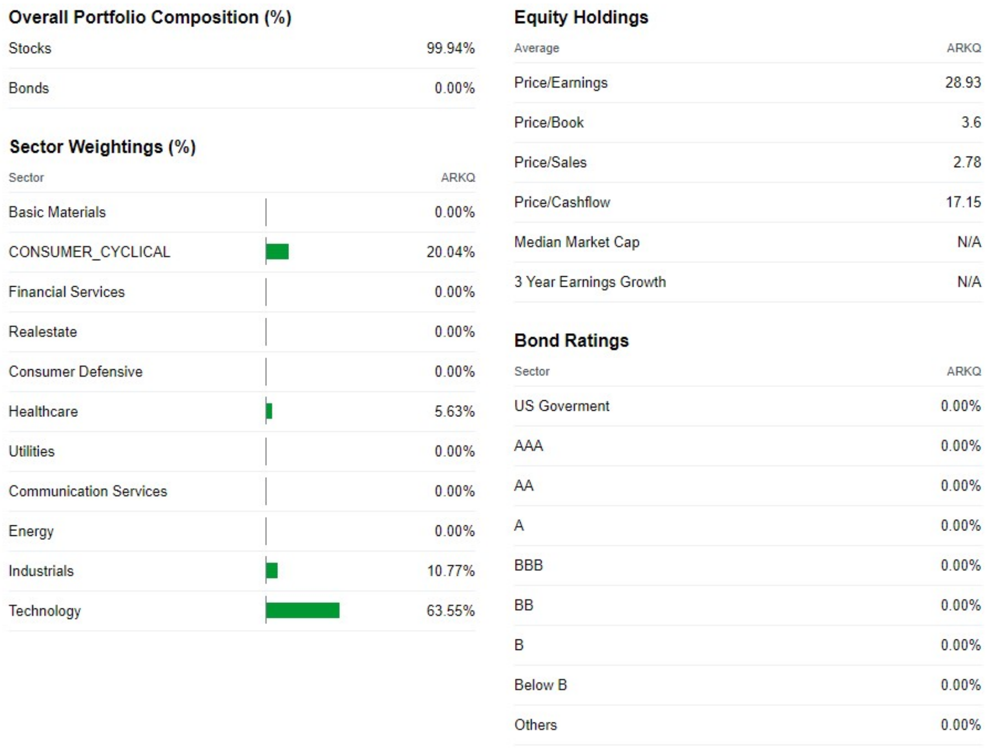
Let's parse the summary page; this will be the easiest because it's stored in a table. Grab all the columns from the table using `find_all('td')` . From here, we can use list comprehension to store all the data, and then add that list to our `major_list` . This will make it easier to break the list into chunks.

In [54]:
```
# SECTION ONE - PARSE THE SUMMARY PAGE

# grab the `tbody` for both the left and right side.
tbody_left = left_summary_table[0].tbody
tbody_right = right_summary_table[0].tbody

# define a list to store both tables
major_list = []

# append the parsed table to the master list.
major_list.append([item.text for item in tbody_left.find_all('td')])
major_list.append([item.text for item in tbody_right.find_all('td')])

# create a chunked version of our master list.
summary_data = [chunk for item in major_list for chunk in split_list(item, 2)]
summary_data

# make it number friendly
for row in summary_data:

    # handle the precentage case
    if '%' in row[1]:
        row[1] = float(row[1].replace('%',''))/100

    # handle the split case X
    elif 'x' in row[1]:
        row[1] = row[1].split(' x ')

    # handle the split case -
    elif '-' in row[1]:
        row[1] = row[1].split(' - ')

    # handle the ,
    elif ',' in row[1]:
        row[1] = float(row[1].replace(',',''))

    # handle missing values
    elif 'N/A' in row[1]:
        row[1] = np.nan
```

```
summary_data
```

```
Out[54]: [['Previous Close', '32.85'],
          ['Open', '32.88'],
          ['Bid', ['33.16', '1200']],
          ['Ask', ['33.21', '1300']],
          ["Day's Range", ['32.78', '33.18']],
          ['52 Week Range', ['28.29', '37.47']],
          ['Volume', 14754.0],
          ['Avg. Volume', 22214.0],
          ['Net Assets', '176.59M'],
          ['NAV', '32.81'],
          ['PE Ratio (TTM)', nan],
          ['Yield', 0.0],
          ['YTD Return', 0.08199999999999999],
          ['Beta (3Y Monthly)', '1.44'],
          ['Expense Ratio (net)', 0.0075],
          ['Inception Date', ['2014-09-30']]]
```

The Holdings page will contain the makeup of the fund; this will be valuable when it comes to determining exposure to different sectors. It appears as the following:



Overall Portfolio Composition (%)

Stocks	99.94%
Bonds	0.00%

Sector Weightings (%)

Sector	ARKQ
Basic Materials	0.00%
CONSUMER_CYCLICAL	20.04%
Financial Services	0.00%
Realestate	0.00%
Consumer Defensive	0.00%
Healthcare	5.63%
Utilities	0.00%
Communication Services	0.00%
Energy	0.00%
Industrials	10.77%
Technology	63.55%

Equity Holdings

Average	ARKQ
Price/Earnings	28.93
Price/Book	3.6
Price/Sales	2.78
Price/Cashflow	17.15
Median Market Cap	N/A
3 Year Earnings Growth	N/A

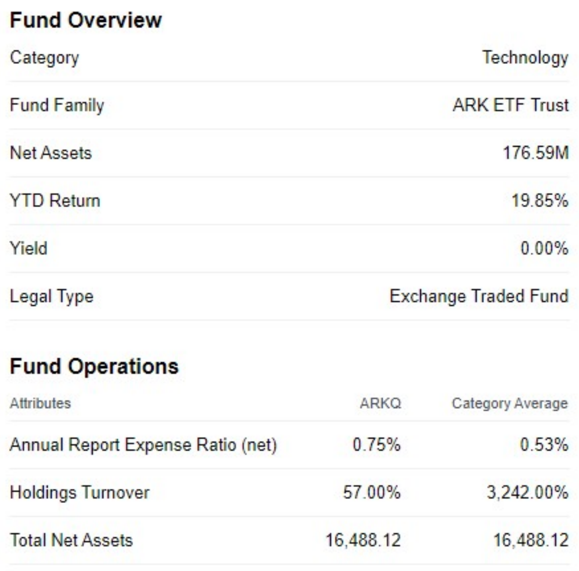
Bond Ratings

Sector	ARKQ
US Goverment	0.00%
AAA	0.00%
AA	0.00%
A	0.00%
BBB	0.00%
BB	0.00%
B	0.00%
Below B	0.00%
Others	0.00%

In [55]:
```python
# SECTION TWO - PARSE THE HOLDING PAGE

# define the link to the page
link = link_dictionary['Holdings']

# request the link and dump the content into the parser.
response = requests.get(link)
soup = BeautifulSoup(response.content, 'html.parser')

# We have to define a list of items we don't want. Luckily there are only a few items we need to avoid.
skip_list = ['',stock_symbol,'Sector','Average']

# find all the span elements labeled with start and end.
items = soup.find_all('span', {'class':['Fl(start)','Fl(end)']})

# loop through the parsed content, grab the text, and make sure it's not in the skip_list.
unsplit_data = [item.text for item in items if item.text not in skip_list ]

# split the list into chunks
holdings_data = [chunk for chunk in split_list(unsplit_data, 2)]

# make number friendly
for row in holdings_data:

    if '%' in row[1]:
        row[1] = float(row[1].replace('%',''))/100

    elif 'N/A' in row[1]:
        row[1] = np.nan

holdings_data
```

```
Out[55]: [['Stocks', 0.9994],
          ['Bonds', 0.0],
          ['Basic Materials', 0.0],
          ['CONSUMER_CYCLICAL', 0.2004],
          ['Financial Services', 0.0],
          ['Realestate', 0.0],
          ['Consumer Defensive', 0.0],
          ['Healthcare', 0.0562999999999996],
          ['Utilities', 0.0],
          ['Communication Services', 0.0],
          ['Energy', 0.0],
          ['Industrials', 0.1076999999999999],
          ['Technology', 0.6355],
          ['Price/Earnings', '28.93'],
          ['Price/Book', '3.6'],
          ['Price/Sales', '2.78'],
          ['Price/Cashflow', '17.15'],
          ['Median Market Cap', nan],
          ['3 Year Earnings Growth', nan],
          ['US Goverment', 0.0],
          ['AAA', 0.0],
          ['AA', 0.0],
          ['A', 0.0],
          ['BBB', 0.0],
          ['BB', 0.0],
          ['B', 0.0],
          ['Below B', 0.0],
          ['Others', 0.0]]
```

The Profile page contains the fund overview and fund operations table it will appear as the following:



Fund Overview

Category	Technology
Fund Family	ARK ETF Trust
Net Assets	176.59M
YTD Return	19.85%
Yield	0.00%
Legal Type	Exchange Traded Fund

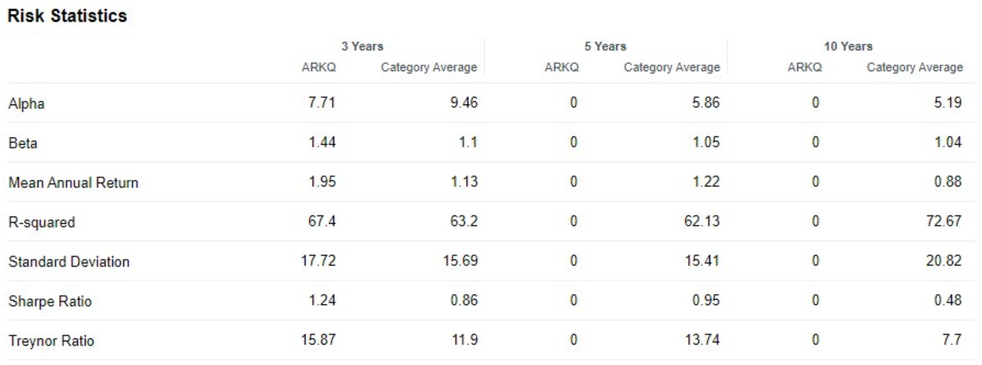
Fund Operations

Attributes	ARKQ	Category Average
Annual Report Expense Ratio (net)	0.75%	0.53%
Holdings Turnover	57.00%	3,242.00%
Total Net Assets	16,488.12	16,488.12

In [61]:
```python
# SECTION THREE - PROFILE PAGE

# define the link
link = link_dictionary['Profile']

# request the link and dump the content into the parser.
response = requests.get(link)
soup = BeautifulSoup(response.content, 'html.parser')

# define a master and mini list
profile_list = []
mini_list = []

# items to skip over.
skip_list = ['',stock_symbol,'Attributes','Category Average']

multi_section = ['Annual Report Expense Ratio (net)','Holdings Turnover','Total Net Assets']

# find all the start and end spans.
for item in soup.find_all('span', {'class':['Fl(start)','Fl(end)']}):

    # if it's not in the skip list, append it.
    if item.text not in skip_list:

        # handle the 3 item section
        if 'Ta(s)' in item['class'] or 'Ta(e)' in item['class']:

            # make number friendly
            if '%' in item.text:
                item =  float(item.text.replace('%','').replace(',',''))/100
            elif ',' in item.text:
                item =  float(item.text.replace(',',''))
            else:
                item = item.text

            mini_list.append(item)

            # once you have two items in the mini_list append to the master list.
            if len(mini_list) == 3:
                profile_list.append(mini_list)
                mini_list = []
        # handle the 2 item section
        else:
            mini_list.append(item.text)

            # once you have two items in the mini_list append to the master list.
            if len(mini_list) == 2:
                profile_list.append(mini_list)
                mini_list = []

print(profile_list)
```

```
[['Category', 'Technology'], ['Fund Family', 'ARK ETF Trust'], ['Net Assets', '176.59M'], ['YTD Return', '8.2
0%'], ['Yield', '0.00%'], ['Legal Type', 'Exchange Traded Fund'], ['Annual Report Expense Ratio (net)', 0.007
5, 0.0053], ['Holdings Turnover', 0.57, 32.42], ['Total Net Assets', 16488.12, 16488.12]]
```

The Risk page contains all the risk metrics we would calculate over different timelines. Here is how the table will appear:



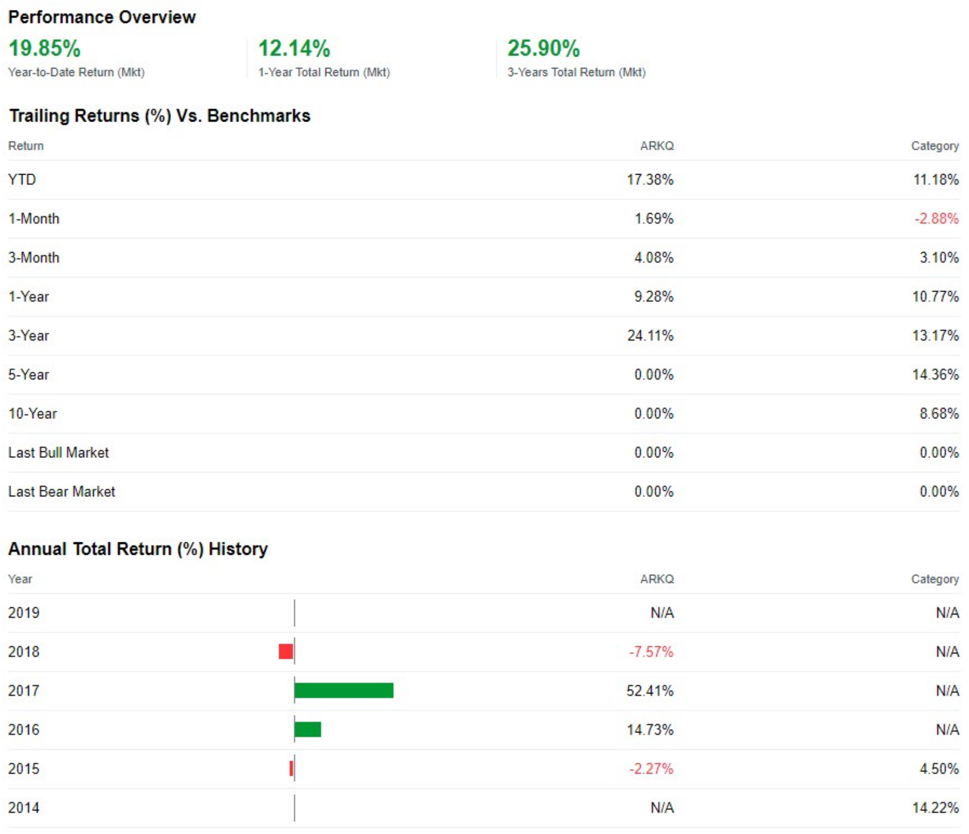
Risk Statistics

	3 Years		5 Years		10 Years	
	ARKQ	Category Average	ARKQ	Category Average	ARKQ	Category Average
Alpha	7.71	9.46	0	5.86	0	5.19
Beta	1.44	1.1	0	1.05	0	1.04
Mean Annual Return	1.95	1.13	0	1.22	0	0.88
R-squared	67.4	63.2	0	62.13	0	72.67
Standard Deviation	17.72	15.69	0	15.41	0	20.82
Sharpe Ratio	1.24	0.86	0	0.95	0	0.48
Treynor Ratio	15.87	11.9	0	13.74	0	7.7

In [64]:
```python
# SECTION THREE - RISK PAGE

# define the link
link = link_dictionary['Risk']

# request the link and dump the content into the parser.
response = requests.get(link)
soup = BeautifulSoup(response.content, 'html.parser')

# define a master and mini list
risk_list = []
mini_list = []

# define the first row as the key row.
risk_list.append(['Category','3-Year Fund','3-Year Category',
                              '5-Year Fund','5-Year Category',
                              '10-Year Fund','10-Year Category'])

# items to skip
list_of_nos = ['',stock_symbol,'Sector','Average']

# find the table
for row in soup.find_all('div', class_= r"Bdbw(1px) Bdbc($screenerBorderGray) Bdbs(s) H(25px) Pt(10px)"):

    # grab the rows
    for item in row.find_all('span'):

        # if it's not in the skip list, append it.
        if item.text not in list_of_nos:

            try:
                mini_list.append(float(item.text))
            except:
                mini_list.append(item.text)

            # chunk it
            if len(mini_list) == 7:
                risk_list.append(mini_list)
                mini_list = []

print(risk_list)
```

```
[['Category', '3-Year Fund', '3-Year Category', '5-Year Fund', '5-Year Category', '10-Year Fund', '10-Year Ca
tegory'], ['Alpha', 7.71, 9.46, 0.0, 5.86, 0.0, 5.19], ['Beta', 1.44, 1.1, 0.0, 1.05, 0.0, 1.04], ['Mean Annu
al Return', 1.95, 1.13, 0.0, 1.22, 0.0, 0.88], ['R-squared', 67.4, 63.2, 0.0, 62.13, 0.0, 72.67], ['Standard
Deviation', 17.72, 15.69, 0.0, 15.41, 0.0, 20.82], ['Sharpe Ratio', 1.24, 0.86, 0.0, 0.95, 0.0, 0.48], ['Trey
nor Ratio', 15.87, 11.9, 0.0, 13.74, 0.0, 7.7]]
```

The performance page, contains all the different return metrics and return benchmarks, it will appear as the following:



Performance Overview

19.85%	12.14%	25.90%
Year-to-Date Return (Mkt)	1-Year Total Return (Mkt)	3-Years Total Return (Mkt)

Trailing Returns (%) Vs. Benchmarks

Return	ARKQ	Category
YTD	17.38%	11.18%
1-Month	1.69%	-2.88%
3-Month	4.08%	3.10%
1-Year	9.28%	10.77%
3-Year	24.11%	13.17%
5-Year	0.00%	14.36%
10-Year	0.00%	8.68%
Last Bull Market	0.00%	0.00%
Last Bear Market	0.00%	0.00%

Annual Total Return (%) History

Year	ARKQ	Category
2019	N/A	N/A
2018	-7.57%	N/A
2017	52.41%	N/A
2016	14.73%	N/A
2015	-2.27%	4.50%
2014	N/A	14.22%

In [66]:
```python
# SECTION FOUR - PERFORMANCE PAGE

# define the link
link = link_dictionary['Performance']

# request the link and dump the content into the parser.
response = requests.get(link)
soup = BeautifulSoup(response.content, 'html.parser')

# define a major and minor list.
performance_list = []
mini_list = []

# define items to skip
skip_list = ['',stock_symbol,'Sector','Average','Performance Overview', 'Return','Category']

# find all three sections, and label them for iteration.
for index, row in enumerate(soup.find_all('div', class_= r"Mb(25px)")):

    # PERFORMANCE OVERVIEW table.
    if index == 0:

        # find all the rows.
        for item in row.find_all('span', class_=True):

            # grab the Metric
            if item.text not in skip_list:

                if '%' in item.text:
                    item =  float(item.text.replace('%',''))/100
                else:
                    item = item.text

                mini_list.append(item)

                # chunk it.
                if len(mini_list) == 2:
                    performance_list.append(mini_list)
                    mini_list = []

    # TRAILING RETURNS AND BENCHMARK table.
    elif index == 1:
        # find all the rows.
        for item in row.find_all('span', class_=True):

            # if it is a metric row  header, nested span tag. Define a row header key
            if item.span != None:
                cat = item.span.text

            # if it's not a metric row header.
            if item.text not in skip_list and item.span == None:

                mini_list.append(float(item.text.replace('%',''))/100)

                if len(mini_list) == 3:
                    mini_list.append(cat)
                    performance_list.append(mini_list)
                    mini_list = []

    # ANNUAL RETURN HISTORY table
    elif index == 2:

        # grab the rows.
        for item in row.find_all('span', class_=True):

            # grab the metric
            if item.text not in skip_list and item.span == None:

                # make number friendly
                if '%' in item.text:
                    item =  float(item.text.replace('%',''))/100

                elif 'N/A' in item.text:
                    item = np.nan

                else:
                    item = item.text

                mini_list.append(item)

                # chunk it.
                if len(mini_list) == 3:
                    performance_list.append(mini_list)
                    mini_list = []
```

```
print(performance_list)
```

```
[[0.08199999999999999, 'Year-to-Date Return (Mkt)'], [-0.05, '1-Year Total Return (Mkt)'], [0.2276999999999999
99, '3-Years Total Return (Mkt)'], [0.17379999999999998, 0.1118, 0.0169, '1-Month'], [-0.0288, 0.0408, 0.031,
'3-Month'], [0.0928, 0.10769999999999999, 0.24109999999999998, '3-Year'], [0.1317, 0.0, 0.1436, '5-Year'],
[0.0, 0.0868, 0.0, 'Last Bull Market'], [0.0, 0.0, 0.0, 'Last Bear Market'], ['2019', nan, nan], ['2018', -0.
0757, nan], ['2017', 0.5241, nan], ['2016', 0.14730000000000001, nan], ['2015', -0.0227, 0.045], ['2014', na
n, 0.1422]]
```