PY0101EN-5.1_Intro_API

August 20, 2020

A pplication Programming Interface (API)

An API lets two pieces of software talk to each other. Just like a function, you don't have to know how the API works only its inputs and outputs. An essential type of API is a REST API that allows you to access resources via the internet. In this lab, we will review the Pandas Library in the context of an API, we will also review a basic REST API

```
<a href="https://cocl.us/topNotebooksPython101Coursera">
     <img src="https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClas
</a>
```

0.1 Table of Contents

Estimated Time Needed: 15 min

[2]:
```
!pip install nba_api
```

```
Collecting nba_api
  Downloading https://files.pythonhosted.org/packages/f0/07/d32f5106c95fbe
e8e54b22d2795f94c2d2213ed6d2e5caac390b56667d37/nba_api-1.1.9-py3-none-any.whl
(242kB)
     |                  | 245kB 6.1MB/s eta 0:00:01
Requirement already satisfied: requests in
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from nba_api)
(2.24.0)
Requirement already satisfied: idna<3,>=2.5 in
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests->nba_api) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests->nba_api) (2020.6.20)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests->nba_api) (1.25.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests->nba_api) (3.0.4)
Installing collected packages: nba-api
Successfully installed nba-api-1.1.9
```

Pandas is an API

You will use this function in the lab:

```
[3]: def one_dict(list_dict):
         keys=list_dict[0].keys()
         out_dict={key:[] for key in keys}
         for dict_ in list_dict:
             for key, value in dict_.items():
                 out_dict[key].append(value)
         return out_dict
```

Pandas is an API

Pandas is actually set of software components , much of witch is not even written in Python.

```
[4]: import pandas as pd
     import matplotlib.pyplot as plt
```

You create a dictionary, this is just data.

```
[5]: dict_={'a':[11,21,31],'b':[12,22,32]}
```

When you create a Pandas object with the Dataframe constructor in API lingo, this is an "instance". The data in the dictionary is passed along to the pandas API. You then use the dataframe to communicate with the API.

```
[6]: df=pd.DataFrame(dict_)
     type(df)
```

```
[6]: pandas.core.frame.DataFrame
```

When you call the method head the dataframe communicates with the API displaying the first few rows of the dataframe.

```
[7]: df.head()
```

```
[7]:     a   b
     0  11  12
     1  21  22
     2  31  32
```

When you call the method mean,the API will calculate the mean and return the value.

```
[8]: df.mean()
```

```
[8]: a    21.0
     b    22.0
     dtype: float64
```

REST APIs

Rest API's function by sending a request, the request is communicated via HTTP message. The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service or resource to perform. In a similar manner, API returns a response, via an HTTP message, this response is usually contained within a JSON.

In this lab, we will use the NBA API to determine how well the Golden State Warriors performed against the Toronto Raptors. We will use the API do the determined number of points the Golden State Warriors won or lost by for each game. So if the value is three, the Golden State Warriors won by three points. Similarly it the Golden State Warriors lost by two points the result will be negative two. The API is reltivly will handle a lot of the details such a Endpoints and Authentication

In the nba api to make a request for a specific team, it's quite simple, we don't require a JSON all we require is an id. This information is stored locally in the API we import the module teams

```
[9]: from nba_api.stats.static import teams
     import matplotlib.pyplot as plt
```

```
[10]: #https://pypi.org/project/nba-api/
```

The method get_teams() returns a list of dictionaries the dictionary key id has a unique identifier for each team as a value

```
[11]: nba_teams = teams.get_teams()
```

The dictionary key id has a unique identifier for each team as a value, let's look at the first three elements of the list:

```
[12]: nba_teams[0:3]
```

```
[12]: [{'id': 1610612737,
        'full_name': 'Atlanta Hawks',
        'abbreviation': 'ATL',
        'nickname': 'Hawks',
        'city': 'Atlanta',
        'state': 'Atlanta',
        'year_founded': 1949},
       {'id': 1610612738,
        'full_name': 'Boston Celtics',
        'abbreviation': 'BOS',
        'nickname': 'Celtics',
        'city': 'Boston',
        'state': 'Massachusetts',
        'year_founded': 1946},
       {'id': 1610612739,
        'full_name': 'Cleveland Cavaliers',
        'abbreviation': 'CLE',
        'nickname': 'Cavaliers',
        'city': 'Cleveland',
        'state': 'Ohio',
        'year_founded': 1970}]
```

To make things easier, we can convert the dictionary to a table. First, we use the function one dict, to create a dictionary. We use the common keys for each team as the keys, the value is a list; each element of the list corresponds to the values for each team. We then convert the dictionary to a dataframe, each row contains the information for a different team.

```
[13]: dict_nba_team=one_dict(nba_teams)
      df_teams=pd.DataFrame(dict_nba_team)
      df_teams.head()
```

```
[13]:          id              full_name abbreviation   nickname         city  \
      0  1610612737          Atlanta Hawks          ATL      Hawks      Atlanta
      1  1610612738         Boston Celtics          BOS    Celtics       Boston
      2  1610612739    Cleveland Cavaliers          CLE  Cavaliers    Cleveland
      3  1610612740   New Orleans Pelicans          NOP   Pelicans  New Orleans
      4  1610612741          Chicago Bulls          CHI      Bulls      Chicago

                 state  year_founded
      0        Atlanta          1949
      1  Massachusetts          1946
      2           Ohio          1970
      3      Louisiana          2002
      4       Illinois          1966
```

Will use the team's nickname to find the unique id, we can see the row that contains the warriors by using the column nickname as follows:

```
[14]: df_warriors=df_teams[df_teams['nickname']=='Warriors']
      df_warriors
```

```
[14]:          id               full_name abbreviation  nickname          city  \
      7  1610612744  Golden State Warriors          GSW  Warriors  Golden State

              state  year_founded
      7  California          1946
```

we can use the following line of code to access the first column of the dataframe:

```
[15]: id_warriors=df_warriors[['id']].values[0][0]
      #we now have an integer that can be used   to request the Warriors information
      id_warriors
```

```
[15]: 1610612744
```

The function “League Game Finder” will make an API call, its in the module stats.endpoints

```
[16]: from nba_api.stats.endpoints import leaguegamefinder
```

The parameter team_id_nullable is the unique ID for the warriors. Under the hood, the NBA API is making a HTTP request.
The information requested is provided and is transmitted via an HTTP response this is assigned to the object gamefinder.

```
[18]: # Since https://stats.nba.com does lot allow api calls from Cloud IPs and␣
       ↪Skills Network Labs uses a Cloud IP.
      # The following code is comment out, you can run it on jupyter labs on your own␣
       ↪computer.
      gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=id_warriors)
```

```
       ␣
↪---------------------------------------------------------------------------

       timeout                                   Traceback (most recent call␣
↪last)

       ~/conda/envs/python/lib/python3.6/site-packages/urllib3/connectionpool.
↪py in _make_request(self, conn, method, url, timeout, chunked,␣
↪**httplib_request_kw)
       425                         # Otherwise it looks like a bug in the code.
   --> 426                         six.raise_from(e, None)
       427             except (SocketTimeout, BaseSSLError, SocketError) as e:


       ~/conda/envs/python/lib/python3.6/site-packages/urllib3/packages/six.py␣
↪in raise_from(value, from_value)


       ~/conda/envs/python/lib/python3.6/site-packages/urllib3/connectionpool.
↪py in _make_request(self, conn, method, url, timeout, chunked,␣
↪**httplib_request_kw)
       420                     try:
   --> 421                         httplib_response = conn.getresponse()
       422                     except BaseException as e:


       ~/conda/envs/python/lib/python3.6/http/client.py in getresponse(self)
      1363                 try:
   -> 1364                     response.begin()
      1365                 except ConnectionError:
```

```
    ~/conda/envs/python/lib/python3.6/http/client.py in begin(self)
    306         while True:
--> 307             version, status, reason = self._read_status()
    308             if status != CONTINUE:


    ~/conda/envs/python/lib/python3.6/http/client.py in _read_status(self)
    267     def _read_status(self):
--> 268         line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
    269         if len(line) > _MAXLINE:


    ~/conda/envs/python/lib/python3.6/socket.py in readinto(self, b)
    585             try:
--> 586                 return self._sock.recv_into(b)
    587             except timeout:


    ~/conda/envs/python/lib/python3.6/ssl.py in recv_into(self, buffer,␣
↪nbytes, flags)
   1011                   self.__class__)
-> 1012             return self.read(nbytes, buffer)
   1013         else:


    ~/conda/envs/python/lib/python3.6/ssl.py in read(self, len, buffer)
    873         try:
--> 874             return self._sslobj.read(len, buffer)
    875         except SSLError as x:


    ~/conda/envs/python/lib/python3.6/ssl.py in read(self, len, buffer)
    630         if buffer is not None:
--> 631             v = self._sslobj.read(len, buffer)
    632         else:


    timeout: The read operation timed out


During handling of the above exception, another exception occurred:


    ReadTimeoutError                          Traceback (most recent call␣
↪last)
```

```
    ~/conda/envs/python/lib/python3.6/site-packages/requests/adapters.py in␣
↪send(self, request, stream, timeout, verify, cert, proxies)
    448                     retries=self.max_retries,
--> 449                     timeout=timeout
    450                 )


    ~/conda/envs/python/lib/python3.6/site-packages/urllib3/connectionpool.
↪py in urlopen(self, method, url, body, headers, retries, redirect,␣
↪assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos,␣
↪**response_kw)
    726             retries = retries.increment(
--> 727                 method, url, error=e, _pool=self, _stacktrace=sys.
↪exc_info()[2]
    728             )


    ~/conda/envs/python/lib/python3.6/site-packages/urllib3/util/retry.py in␣
↪increment(self, method, url, response, error, _pool, _stacktrace)
    402             if read is False or not self.
↪_is_method_retryable(method):
--> 403                 raise six.reraise(type(error), error, _stacktrace)
    404             elif read is not None:


    ~/conda/envs/python/lib/python3.6/site-packages/urllib3/packages/six.py␣
↪in reraise(tp, value, tb)
    734                 raise value.with_traceback(tb)
--> 735             raise value
    736         finally:


    ~/conda/envs/python/lib/python3.6/site-packages/urllib3/connectionpool.
↪py in urlopen(self, method, url, body, headers, retries, redirect,␣
↪assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos,␣
↪**response_kw)
    676                 headers=headers,
--> 677                 chunked=chunked,
    678             )


    ~/conda/envs/python/lib/python3.6/site-packages/urllib3/connectionpool.
↪py in _make_request(self, conn, method, url, timeout, chunked,␣
↪**httplib_request_kw)
    427         except (SocketTimeout, BaseSSLError, SocketError) as e:
```

```
   --> 428             self._raise_timeout(err=e, url=url,␣
↪timeout_value=read_timeout)
       429             raise


       ~/conda/envs/python/lib/python3.6/site-packages/urllib3/connectionpool.
↪py in _raise_timeout(self, err, url, timeout_value)
       335             raise ReadTimeoutError(
   --> 336                 self, url, "Read timed out. (read timeout=%s)" %␣
↪timeout_value
       337             )


       ReadTimeoutError: HTTPSConnectionPool(host='stats.nba.com', port=443):␣
↪Read timed out. (read timeout=30)


   During handling of the above exception, another exception occurred:


       ReadTimeout                               Traceback (most recent call␣
↪last)

       <ipython-input-18-c455b38b5f7c> in <module>
         1 # Since https://stats.nba.com does lot allow api calls from Cloud␣
↪IPs and Skills Network Labs uses a Cloud IP.
         2 # The following code is comment out, you can run it on jupyter labs␣
↪on your own computer.
   ----> 3 gamefinder = leaguegamefinder.
↪LeagueGameFinder(team_id_nullable=id_warriors)
```

```
       ~/conda/envs/python/lib/python3.6/site-packages/nba_api/stats/endpoints/
↪leaguegamefinder.py in __init__(self, player_or_team_abbreviation,␣
↪conference_nullable, date_from_nullable, date_to_nullable,␣
↪division_simple_nullable, draft_number_nullable, draft_round_nullable,␣
↪draft_team_id_nullable, draft_year_nullable, eq_ast_nullable, eq_blk_nullable,␣
↪eq_dd_nullable, eq_dreb_nullable, eq_fg3a_nullable, eq_fg3m_nullable,␣
↪eq_fg3_pct_nullable, eq_fga_nullable, eq_fgm_nullable, eq_fg_pct_nullable,␣
↪eq_fta_nullable, eq_ftm_nullable, eq_ft_pct_nullable, eq_minutes_nullable,␣
↪eq_oreb_nullable, eq_pf_nullable, eq_pts_nullable, eq_reb_nullable,␣
↪eq_stl_nullable, eq_td_nullable, eq_tov_nullable, game_id_nullable,␣
↪gt_ast_nullable, gt_blk_nullable, gt_dd_nullable, gt_dreb_nullable,␣
↪gt_fg3a_nullable, gt_fg3m_nullable, gt_fg3_pct_nullable, gt_fga_nullable,␣
↪gt_fgm_nullable, gt_fg_pct_nullable, gt_fta_nullable, gt_ftm_nullable,␣
↪gt_ft_pct_nullable, gt_minutes_nullable, gt_oreb_nullable, gt_pf_nullable,␣
↪gt_pts_nullable, gt_reb_nullable, gt_stl_nullable, gt_td_nullable,␣
↪gt_tov_nullable, league_id_nullable, location_nullable, lt_ast_nullable,␣
↪lt_blk_nullable, lt_dd_nullable, lt_dreb_nullable, lt_fg3a_nullable,␣
↪lt_fg3m_nullable, lt_fg3_pct_nullable, lt_fga_nullable, lt_fgm_nullable,␣
↪lt_fg_pct_nullable, lt_fta_nullable, lt_ftm_nullable, lt_ft_pct_nullable,␣
↪lt_minutes_nullable, lt_oreb_nullable, lt_pf_nullable, lt_pts_nullable,␣
↪lt_reb_nullable, lt_stl_nullable, lt_td_nullable, lt_tov_nullable,␣
↪outcome_nullable, po_round_nullable, player_id_nullable, rookie_year_nullable,␣
↪season_nullable, season_segment_nullable, season_type_nullable,␣
↪starter_bench_nullable, team_id_nullable, vs_conference_nullable,␣
↪vs_division_nullable, vs_team_id_nullable, years_experience_nullable, proxy,␣
↪headers, timeout, get_request)
       202         }
       203         if get_request:
   --> 204             self.get_request()
       205
       206     def get_request(self):


       ~/conda/envs/python/lib/python3.6/site-packages/nba_api/stats/endpoints/
↪leaguegamefinder.py in get_request(self)
       210             proxy=self.proxy,
       211             headers=self.headers,
   --> 212             timeout=self.timeout,
       213         )
       214         self.load_response()


       ~/conda/envs/python/lib/python3.6/site-packages/nba_api/library/http.py␣
↪in send_api_request(self, endpoint, parameters, referer, proxy, headers,␣
↪timeout, raise_exception_on_error)
       128
       129         if not contents:
```

```
  --> 130             response = requests.get(url=base_url, params=parameters,␣
↪headers=request_headers, proxies=proxies, timeout=timeout)
      131             url = response.url
      132             status_code = response.status_code


      ~/conda/envs/python/lib/python3.6/site-packages/requests/api.py in␣
↪get(url, params, **kwargs)
       74
       75     kwargs.setdefault('allow_redirects', True)
  ---> 76     return request('get', url, params=params, **kwargs)
       77
       78


      ~/conda/envs/python/lib/python3.6/site-packages/requests/api.py in␣
↪request(method, url, **kwargs)
       59     # cases, and look like a memory leak in others.
       60     with sessions.Session() as session:
  ---> 61         return session.request(method=method, url=url, **kwargs)
       62
       63


      ~/conda/envs/python/lib/python3.6/site-packages/requests/sessions.py in␣
↪request(self, method, url, params, data, headers, cookies, files, auth,␣
↪timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
      528         }
      529         send_kwargs.update(settings)
  --> 530         resp = self.send(prep, **send_kwargs)
      531
      532         return resp


      ~/conda/envs/python/lib/python3.6/site-packages/requests/sessions.py in␣
↪send(self, request, **kwargs)
      641
      642         # Send the request
  --> 643         r = adapter.send(request, **kwargs)
      644
      645         # Total elapsed time of the request (approximately)


      ~/conda/envs/python/lib/python3.6/site-packages/requests/adapters.py in␣
↪send(self, request, stream, timeout, verify, cert, proxies)
      527                 raise SSLError(e, request=request)
      528             elif isinstance(e, ReadTimeoutError):
```

```
--> 529                 raise ReadTimeout(e, request=request)
    530             else:
    531                 raise


    ReadTimeout: HTTPSConnectionPool(host='stats.nba.com', port=443): Read␣
↪timed out. (read timeout=30)
```

we can see the json file by running the following line of code.

[19]:
```
# Since https://stats.nba.com does lot allow api calls from Cloud IPs and␣
↪Skills Network Labs uses a Cloud IP.
# The following code is comment out, you can run it on jupyter labs on your own␣
↪computer.
# gamefinder.get_json()
```

The game finder object has a method get_data_frames(), that returns a dataframe. If we view the dataframe, we can see it contains information about all the games the Warriors played. The PLUS_MINUS column contains information on the score, if the value is negative the Warriors lost by that many points, if the value is positive, the warriors one by that amount of points. The column MATCHUP had the team the Warriors were playing, GSW stands for golden state and TOR means Toronto Raptors; vs signifies it was a home game and the @ symbol means an away game.

[20]:
```
# Since https://stats.nba.com does lot allow api calls from Cloud IPs and␣
↪Skills Network Labs uses a Cloud IP.
# The following code is comment out, you can run it on jupyter labs on your own␣
↪computer.
# games = gamefinder.get_data_frames()[0]
# games.head()
```

you can download the dataframe from the API call for Golden State and run the rest like a video.

[21]:
```
! wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/
↪CognitiveClass/PY0101EN/Chapter%205/Labs/Golden_State.pkl
```

```
--2020-08-20 14:12:48--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-
courses-data/CognitiveClass/PY0101EN/Chapter%205/Labs/Golden_State.pkl
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)… 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)|67.228.254.196|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 811065 (792K) [application/octet-stream]
Saving to: ‘Golden_State.pkl’

Golden_State.pkl    100%[===================>] 792.06K  2.44MB/s    in 0.3s
```

```
2020-08-20 14:12:49 (2.44 MB/s) - ‘Golden_State.pkl’ saved [811065/811065]
```

```
[22]: file_name = "Golden_State.pkl"
      games = pd.read_pickle(file_name)
      games.head()
```

```
[22]:   SEASON_ID     TEAM_ID TEAM_ABBREVIATION              TEAM_NAME     GAME_ID  \
      0     22019  1610612744               GSW  Golden State Warriors  1521900066
      1     22019  1610612744               GSW  Golden State Warriors  1521900058
      2     22019  1610612744               GSW  Golden State Warriors  1521900039
      3     22019  1610612744               GSW  Golden State Warriors  1521900020
      4     22019  1610612744               GSW  Golden State Warriors  1521900007

          GAME_DATE      MATCHUP WL  MIN  PTS  …  FT_PCT  OREB  DREB   REB  AST  \
      0  2019-07-12  GSW vs. LAL  L  200   87  …   0.800  13.0  29.0  42.0   13
      1  2019-07-10    GSW @ DEN  W  201   73  …   0.867   7.0  27.0  34.0   10
      2  2019-07-08    GSW @ LAL  W  200   88  …   0.621   8.0  29.0  37.0   21
      3  2019-07-07  GSW vs. TOR  W  201   80  …   0.923   6.0  37.0  43.0   18
      4  2019-07-05  GSW vs. CHA  L  200   85  …   0.889   8.0  28.0  36.0   19

          STL  BLK   TOV  PF  PLUS_MINUS
      0  10.0    3  11.0  21         3.2
      1  11.0    7  20.0  20        -8.0
      2  10.0    4  13.0  22         8.0
      3   8.0    3  20.0  25        10.0
      4   9.0    3  13.0  15        -8.0

      [5 rows x 28 columns]
```

We can create two dataframes, one for the games that the Warriors faced the raptors at home and the second for away games.

```
[23]: games_home=games [games ['MATCHUP']=='GSW vs. TOR']
      games_away=games [games ['MATCHUP']=='GSW @ TOR']
```

We can calculate the mean for the column PLUS_MINUS for the dataframes games_home and games_away:

```
[24]: games_home.mean()['PLUS_MINUS']
```

```
[24]: 3.730769230769231
```

```
[25]: games_away.mean()['PLUS_MINUS']
```

```
[25]: -0.6071428571428571
```

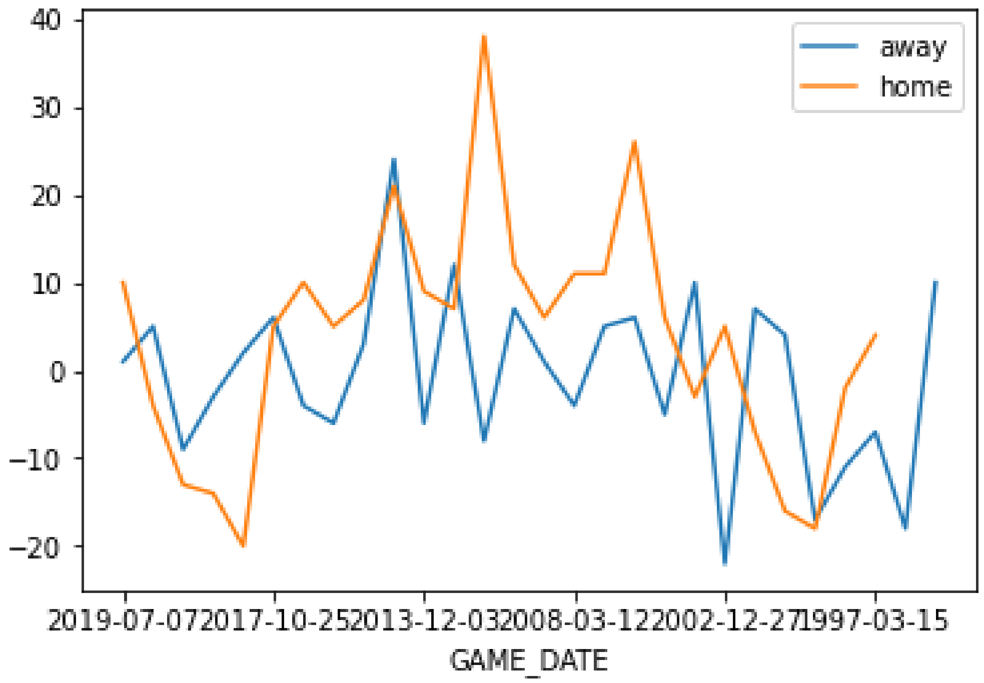
We can plot out the PLUS MINUS column for for the dataframes games_home and games_away.

We see the warriors played better at home.

[26]:
```
fig, ax = plt.subplots()

games_away.plot(x='GAME_DATE',y='PLUS_MINUS', ax=ax)
games_home.plot(x='GAME_DATE',y='PLUS_MINUS', ax=ax)
ax.legend(["away", "home"])
plt.show()
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/pandas/plotting/_matplotlib/core.py:1192: UserWarning: FixedFormatter
should only be used together with FixedLocator
  ax.set_xticklabels(xticklabels)
```



About the Authors: Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.