Segmenting And Clustering Neighborhoods in Toronto Part 3 (Clustering)

August 27, 2019

Import libraries

[58]:
```
import pandas as pd
import numpy as np
import requests
import warnings
warnings.filterwarnings('ignore')
import geopy
from geopy.geocoders import Nominatim
import geocoder
import folium
from sklearn.cluster import KMeans
import matplotlib.cm as cm
import matplotlib.colors as colors
from IPython.display import HTML, display
```

Set the url to table location and get the content of the page in variable

[2]:
```
url = 'https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_M'
page = requests.get(url).content
```

Use the pandas read_html function to read the table. In this case the first table on the page ([0])

[3]:
```
df_raw = pd.read_html(page,header=0)[0]
```

[4]:
```
df_raw.head()
```

[4]:
```
  Postcode           Borough     Neighbourhood
0      M1A      Not assigned      Not assigned
1      M2A      Not assigned      Not assigned
2      M3A        North York         Parkwoods
3      M4A        North York  Victoria Village
4      M5A  Downtown Toronto      Harbourfront
```

Let's clean up the dataset according to specifications, filtering and replacing values

```
[5]: df = df_raw[df_raw['Borough'] != 'Not assigned'] #filter
df.columns = ['PostalCode','Borough','Neighborhood'] #set columns
df.loc[df['Neighborhood'] == 'Not assigned',['Neighborhood']] = df['Borough']␣
 ↪#replace Neigborhood with Borough if Neigborhood = 'Not assigned'
```

Group the dataframe and apply the string concatenation

```
[6]: df = df.groupby(by=['PostalCode','Borough']).agg(lambda x: ', '.join(set(x))).
 ↪reset_index()
```

Finally let's show the final frame (103 records with 3 columns)

```
[7]: df.shape

[7]: (103, 3)
```

We have the dataframe setup, now let's find the longitude and latitude. I'm using the provided csv as the geocoder is malfunctioning

```
[8]: file = 'Geospatial_Coordinates.csv'
df_geo = pd.read_csv(file)
df_geo.rename(columns = {'Postal Code':'PostalCode'}, inplace = True)
```

We now have the location data per postalcode so now it will be joined together with the Toronto dataframe

```
[9]: df_toronto = pd.merge(df,df_geo, on='PostalCode',how='left')
```

This leaves with clean appended dataframe incl lat and long

```
[10]: df_toronto.head()

[10]:   PostalCode      Borough                            Neighborhood   Latitude  \
0          M1B  Scarborough                          Rouge, Malvern  43.806686
1          M1C  Scarborough  Rouge Hill, Highland Creek, Port Union  43.784535
2          M1E  Scarborough       Morningside, West Hill, Guildwood  43.763573
3          M1G  Scarborough                                  Woburn  43.770992
4          M1H  Scarborough                               Cedarbrae  43.773136

   Longitude
0 -79.194353
1 -79.160497
2 -79.188711
3 -79.216917
4 -79.239476
```

Let's start with our first map and plot the locations on the Toronto map

We'll define the start zoom location of the folium map with the address and geolocator to get the latitude and longitude

[11]:
```
address = 'Toronto, Ontario'

geolocator = Nominatim(user_agent="ny_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinates of Toronto are {}, {}.'.format(latitude,␣
 ↪longitude))
```

```
The geograpical coordinates of Toronto are 43.653963, -79.387207.
```

Now we plot the areas onto the Toronto map

[51]:
```
# Function required to show folium maps inline
```

[59]:
```
# create map of New York using latitude and longitude values
map_toronto = folium.Map(location=[latitude, longitude], zoom_start=10)

# add markers to map
for lat, lng, borough, neighborhood in zip(df_toronto['Latitude'],␣
 ↪df_toronto['Longitude'], df_toronto['Borough'], df_toronto['Neighborhood']):
    label = '{}, {}'.format(neighborhood, borough)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=True).add_to(map_toronto)

display(map_toronto)
```

```
<folium.folium.Map at 0x7f99b043bb38>
```

Now we have the initial map setup we continue with clustering. We will start with limiting the dataset as now we have 103 points on the map. We'll limit the set to only show locations for Borough's with the name Toronto in it

[13]:
```
toronto_data = df_toronto[df_toronto.Borough.str.contains('Toronto')].
 ↪reset_index(drop=True)
toronto_data.shape
```

[13]:
```
(38, 5)
```

This leaves us with 38 Boroughs we wil continue to work with. Let's map them again

[37]:
```
# create map of New York using latitude and longitude values
map_toronto_filtered = folium.Map(location=[latitude, longitude], zoom_start=11)

# add markers to map
for lat, lng, borough, neighborhood in zip(toronto_data['Latitude'],␣
 ↪toronto_data['Longitude'], toronto_data['Borough'],␣
 ↪toronto_data['Neighborhood']):
    label = '{}, {}'.format(neighborhood, borough)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_toronto_filtered)

map_toronto_filtered
```

[37]:
```
<folium.folium.Map at 0x7f99b8367710>
```

Let's look at venues with foursquare to the locations we have

[15]:
```
# @hidden cell
CLIENT_ID = '5LAMV3DNDHBER3VMUROMJNYRCJ5S35VSIB5BTJKJW2KHVG55' # your␣
 ↪Foursquare ID
CLIENT_SECRET = '3D5FLKFOOA41T5XPDDIZTLLWTPIJWAMVQIU3AS5POKUEV1BW' # your␣
 ↪Foursquare Secret
VERSION = '20180605' # Foursquare API version
```

[16]:
```
# some variables we need for the below functions

LIMIT = 100 # limit of number of venues returned by Foursquare API
radius = 500 # define radius
```

Function to get venues to process all the neighborhoods in Toronto

[17]:
```
def getNearbyVenues(names, latitudes, longitudes, radius=500):

    venues_list=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)

        # create the API request URL
```

```
        url = 'https://api.foursquare.com/v2/venues/explore?
↪&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            lat,
            lng,
            radius,
            LIMIT)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']

        # return only relevant information for each nearby venue
        venues_list.append([(
            name,
            lat,
            lng,
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item␣
↪in venue_list])
    nearby_venues.columns = ['Neighborhood',
                  'Neighborhood Latitude',
                  'Neighborhood Longitude',
                  'Venue',
                  'Venue Latitude',
                  'Venue Longitude',
                  'Venue Category']

    return(nearby_venues)
```

[18]:
```
#Apply the function
toronto_venues = getNearbyVenues(names=toronto_data ['Neighborhood'],
                                   latitudes=toronto_data ['Latitude'],
                                   longitudes=toronto_data['Longitude'])
```

```
The Beaches
Riverdale, The Danforth West
The Beaches West, India Bazaar
Studio District
Lawrence Park
Davisville North
North Toronto West
Davisville
```

```
Summerhill East, Moore Park
Forest Hill SE, Rathnelly, South Hill, Summerhill West, Deer Park
Rosedale
St. James Town, Cabbagetown
Church and Wellesley
Harbourfront, Regent Park
Garden District, Ryerson
St. James Town
Berczy Park
Central Bay Street
Adelaide, Richmond, King
Harbourfront East, Union Station, Toronto Islands
Toronto Dominion Centre, Design Exchange
Victoria Hotel, Commerce Court
Roselawn
Forest Hill North, Forest Hill West
Yorkville, The Annex, North Midtown
Harbord, University of Toronto
Kensington Market, Grange Park, Chinatown
Island airport, South Niagara, CN Tower, Railway Lands, Harbourfront West,
Bathurst Quay, King and Spadina
Stn A PO Boxes 25 The Esplanade
First Canadian Place, Underground city
Christie
Dovercourt Village, Dufferin
Trinity, Little Portugal
Brockton, Parkdale Village, Exhibition Place
High Park, The Junction South
Roncesvalles, Parkdale
Runnymede, Swansea
Business Reply Mail Processing Centre 969 Eastern
```

So we got 1700 records returned with 7 columns

[19]:
```
print(toronto_venues.shape)
toronto_venues.head()
```

```
(1705, 7)
```

[19]:
```
                   Neighborhood  Neighborhood Latitude  \
0                   The Beaches              43.676357
1                   The Beaches              43.676357
2                   The Beaches              43.676357
3                   The Beaches              43.676357
4  Riverdale, The Danforth West              43.679557

   Neighborhood Longitude                              Venue  Venue Latitude  \
0              -79.293031                   Glen Manor Ravine       43.676821
```

```
1           -79.293031  The Big Carrot Natural Food Market       43.678879
2           -79.293031              Grover Pub and Grub       43.679181
3           -79.293031                    Upper Beaches       43.680563
4           -79.352188                         Pantheon       43.677621

   Venue Longitude     Venue Category
0       -79.293942              Trail
1       -79.297734  Health Food Store
2       -79.297215                Pub
3       -79.292869       Neighborhood
4       -79.351434   Greek Restaurant
```

We continue to analyse the neighborhoods

[20]:
```python
# one hot encoding
toronto_onehot = pd.get_dummies(toronto_venues[['Venue Category']], prefix="",␣
↪prefix_sep="")

# add neighborhood column back to dataframe
toronto_onehot['Neighborhood'] = toronto_venues['Neighborhood']

# move neighborhood column to the first column
fixed_columns = [toronto_onehot.columns[-1]] + list(toronto_onehot.columns[:-1])
toronto_onehot = toronto_onehot[fixed_columns]

toronto_onehot.head()
```

[20]:
```
   Yoga Studio  Afghan Restaurant  Airport  Airport Food Court  Airport Gate  \
0            0                  0        0                   0             0
1            0                  0        0                   0             0
2            0                  0        0                   0             0
3            0                  0        0                   0             0
4            0                  0        0                   0             0

   Airport Lounge  Airport Service  Airport Terminal  American Restaurant  \
0               0                0                 0                    0
1               0                0                 0                    0
2               0                0                 0                    0
3               0                0                 0                    0
4               0                0                 0                    0

   Antique Shop  ...  Theme Restaurant  Thrift / Vintage Store  \
0             0  ...                 0                       0
1             0  ...                 0                       0
2             0  ...                 0                       0
3             0  ...                 0                       0
4             0  ...                 0                       0
```

```
   Toy / Game Store  Trail  Train Station  Vegetarian / Vegan Restaurant  \
0                 0      1              0                              0
1                 0      0              0                              0
2                 0      0              0                              0
3                 0      0              0                              0
4                 0      0              0                              0

   Video Game Store  Vietnamese Restaurant  Wine Bar  Wings Joint
0                 0                      0         0            0
1                 0                      0         0            0
2                 0                      0         0            0
3                 0                      0         0            0
4                 0                      0         0            0

[5 rows x 234 columns]
```

[21]:
```
toronto_onehot.shape
```

[21]:
```
(1705, 234)
```

So got the categories converted to numerical values and transposed them into columns. We have 1700 records with 234 Venues categories

We group them by Neigborhood and that will leave us with 38 neighborhoods with 234 venue categories

[22]:
```
toronto_grouped = toronto_onehot.groupby('Neighborhood').mean().reset_index()
toronto_grouped.shape
```

[22]:
```
(38, 234)
```

That is a lot to process so we will get the top 10 venues for each neighborhood

[23]:
```
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)
    return row_categories_sorted.index.values[0:num_top_venues]
```

[24]:
```
num_top_venues = 10

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['Neighborhood']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
```

```
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = toronto_grouped['Neighborhood']

for ind in np.arange(toronto_grouped.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] =␣
 ↪return_most_common_venues(toronto_grouped.iloc[ind, :], num_top_venues)

neighborhoods_venues_sorted.head(3)
```

[24]:

```
                                   Neighborhood 1st Most Common Venue  \
0                         Adelaide, Richmond, King           Coffee Shop
1                                      Berczy Park           Coffee Shop
2  Brockton, Parkdale Village, Exhibition Place           Coffee Shop

  2nd Most Common Venue 3rd Most Common Venue 4th Most Common Venue  \
0                  Café                   Bar            Steakhouse
1          Cocktail Bar        Farmers Market              Beer Bar
2                  Café        Breakfast Spot         Grocery Store

  5th Most Common Venue 6th Most Common Venue 7th Most Common Venue  \
0        Cosmetics Shop       Thai Restaurant            Restaurant
1                Bakery            Steakhouse           Cheese Shop
2          Intersection     Convenience Store             Pet Store

  8th Most Common Venue 9th Most Common Venue 10th Most Common Venue
0                 Hotel          Burger Joint        Asian Restaurant
1                  Café    Seafood Restaurant      Italian Restaurant
2                   Gym           Climbing Gym    Caribbean Restaurant
```

We will now cluster the neighborhood with *k*-means into 5 clusters

[25]:

```
# set number of clusters
kclusters = 5

toronto_grouped_clustering = toronto_grouped.drop('Neighborhood', 1)

# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=0).
 ↪fit(toronto_grouped_clustering)

# check cluster labels generated for each row in the dataframe
kmeans.labels_[0:10]
```

[25]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

[26]:
```
kmeans_labels = kmeans.labels_
```

Let's add the clusters back to the neighborhoods and venues

[27]:
```
# add clustering labels
neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans_labels)

toronto_merged = toronto_data

# merge toronto_grouped with toronto_data to add latitude/longitude for each␣
↪neighborhood
toronto_merged = toronto_merged.join(neighborhoods_venues_sorted.
↪set_index('Neighborhood'), on='Neighborhood')

toronto_merged.head()
```

[27]:
```
  PostalCode          Borough                      Neighborhood   Latitude  \
0        M4E     East Toronto                       The Beaches  43.676357
1        M4K     East Toronto       Riverdale, The Danforth West  43.679557
2        M4L     East Toronto  The Beaches West, India Bazaar  43.668999
3        M4M     East Toronto                   Studio District  43.659526
4        M4N  Central Toronto                     Lawrence Park  43.728020

   Longitude  Cluster Labels 1st Most Common Venue 2nd Most Common Venue  \
0 -79.293031               0     Health Food Store                 Trail
1 -79.352188               0      Greek Restaurant           Coffee Shop
2 -79.315572               0                  Park         Movie Theater
3 -79.340923               0                  Café           Coffee Shop
4 -79.388790               4                  Park         Jewelry Store

  3rd Most Common Venue   4th Most Common Venue 5th Most Common Venue  \
0                   Pub            Dessert Shop     Falafel Restaurant
1    Italian Restaurant  Furniture / Home Store         Ice Cream Shop
2          Liquor Store              Board Shop         Sandwich Place
3   American Restaurant      Italian Restaurant                 Bakery
4           Swim School                Bus Line            Wings Joint

  6th Most Common Venue      7th Most Common Venue 8th Most Common Venue  \
0           Event Space       Ethiopian Restaurant     Electronics Store
1  Caribbean Restaurant                  Bookstore               Brewery
2          Burger Joint       Fast Food Restaurant         Burrito Place
3    Seafood Restaurant  Latin American Restaurant       Coworking Space
4        Discount Store         Falafel Restaurant           Event Space

         9th Most Common Venue 10th Most Common Venue
0  Eastern European Restaurant     Dumpling Restaurant
1                Bubble Tea Shop           Burger Joint
```

```
2           Fish & Chips Shop            Steakhouse
3                   Bookstore                 Diner
4        Ethiopian Restaurant      Electronics Store
```

Finally, let's visualize the resulting clusters

[28]:
```python
# create map
map_clusters = folium.Map(location=[latitude, longitude], zoom_start=12)

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(toronto_merged['Latitude'],␣
 ↪toronto_merged['Longitude'], toronto_merged['Neighborhood'],␣
 ↪toronto_merged['Cluster Labels']):
    label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=5,
        popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=0.7).add_to(map_clusters)

map_clusters
```

[28]: `<folium.folium.Map at 0x7f99b995cfd0>`

The question then is what do the clusters represent. What is in those various clusters so we can name them better than Cluster 0-4

We will filter the toronto_merged frame into their respective variable so we analyse further

[29]:
```python
label_0 = toronto_merged.loc[toronto_merged['Cluster Labels'] == 0,␣
 ↪toronto_merged.columns[[1] + list(range(5, toronto_merged.shape[1]))]]
label_1 = toronto_merged.loc[toronto_merged['Cluster Labels'] == 1,␣
 ↪toronto_merged.columns[[1] + list(range(5, toronto_merged.shape[1]))]]
label_2 = toronto_merged.loc[toronto_merged['Cluster Labels'] == 2,␣
 ↪toronto_merged.columns[[1] + list(range(5, toronto_merged.shape[1]))]]
label_3 = toronto_merged.loc[toronto_merged['Cluster Labels'] == 3,␣
 ↪toronto_merged.columns[[1] + list(range(5, toronto_merged.shape[1]))]]
```

```
label_4 = toronto_merged.loc[toronto_merged['Cluster Labels'] == 4,␣
↪toronto_merged.columns[[1] + list(range(5, toronto_merged.shape[1]))]]
```

I'm not exactly sure how to find the common ground in the various clusters

[30]:
```
venues_columns = neighborhoods_venues_sorted.columns
venues_columns = venues_columns.drop(['Cluster Labels','Neighborhood'])
```

[39]:
```
label_1.head()
```

[39]:
```
            Borough  Cluster Labels 1st Most Common Venue  \
22  Central Toronto               1                Garden

   2nd Most Common Venue 3rd Most Common Venue 4th Most Common Venue  \
22          Dessert Shop    Falafel Restaurant           Event Space

   5th Most Common Venue 6th Most Common Venue       7th Most Common Venue  \
22  Ethiopian Restaurant     Electronics Store  Eastern European Restaurant

   8th Most Common Venue 9th Most Common Venue 10th Most Common Venue
22   Dumpling Restaurant            Donut Shop       Doner Restaurant
```

[40]:
```
label_4 # park
```

[40]:
```
           Borough  Cluster Labels 1st Most Common Venue  \
4  Central Toronto               4                  Park

  2nd Most Common Venue 3rd Most Common Venue 4th Most Common Venue  \
4         Jewelry Store           Swim School              Bus Line

  5th Most Common Venue 6th Most Common Venue 7th Most Common Venue  \
4           Wings Joint        Discount Store    Falafel Restaurant

  8th Most Common Venue 9th Most Common Venue 10th Most Common Venue
4           Event Space  Ethiopian Restaurant      Electronics Store
```

[46]:
```
file = 'Segmenting And Clustering Neighborhoods in Toronto Part 3 (Clustering).
↪ipynb'
!jupyter nbconvert file --to pdf

!jupyter nbconvert Decorators.ipynb --to html
```

```
[NbConvertApp] WARNING | pattern 'file' matched no files
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------
```

```
Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--debug
    set log level to logging.DEBUG (maximize logging output)
--generate-config
    generate default config file
-y
    Answer yes to any questions instead of prompting.
--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
--stdout
    Write notebook output to stdout instead of files.
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--no-prompt
    Exclude input and output prompts from converted document.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL')
    Set the log level by value or name.
--config=<Unicode> (JupyterApp.config_file)
    Default: ''
    Full path of a config file.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'html_ch', 'html_embed', 'html_toc',
    'html_with_lenvs', 'html_with_toclenvs', 'latex', 'latex_with_lenvs',
    'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'selectLanguage',
    'slides', 'slides_with_lenvs'] or a dotted object name that represents the
```

```
    import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: ''
    Name of the template file to use
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: ''
    PostProcessor class used to write the results of the conversion
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: ''
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy  of reveal.js.
    For speaker notes to work, this must be a relative path to a local  copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
    slideshow) for more details.
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.

To see all available configurables, use `--help-all`

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb

    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'html_ch', 'html_embed',
```

```
'html_toc', 'html_with_lenvs', 'html_with_toclenvs', 'latex',
'latex_with_lenvs', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script',
'selectLanguage', 'slides', 'slides_with_lenvs'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple of
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containing::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
```

[ ]:

```

```