rebrick_04

October 17, 2020

0.1 Example of how to use the rebrick python wrapper for rebrickable api

[92]:
```
import rebrick
import json

# init rebrick module for general reading
rebrick.init("1c3972e38c0479079cfafb4c80b6e11d")

# get set info
response = rebrick.lego.get_set(6608)
print(json.loads(response.read()))
```

```
{'set_num': '6608-1', 'name': 'Tractor', 'year': 1982, 'theme_id': 73,
'num_parts': 21, 'set_img_url':
'https://cdn.rebrickable.com/media/sets/6608-1/9875.jpg', 'set_url':
'https://rebrickable.com/sets/6608-1/tractor/', 'last_modified_dt':
'2015-11-02T10:43:45.980214Z'}
```

[93]:
```
# init rebrick module including user reading
rebrick.init("your_API_KEY_here", "your_USER_TOKEN_here")

# if you don't know the user token you can use your login credentials
rebrick.init("1c3972e38c0479079cfafb4c80b6e11d", "rpmoo", "mcH.3~.EsV;hs*qd")

# get user partlists
response = rebrick.users.get_partlists()
print(json.loads(response.read()))
```

```
{'count': 0, 'next': None, 'previous': None, 'results': []}
```

1 1. Load the latest data sets from rebrickable website

First scrape www.Rebrickable.com for the links we want

[94]:
```
#https://pythonspot.com/extract-links-from-webpage-beautifulsoup/
import os
import requests
from bs4 import BeautifulSoup
```

```
import re
import shutil
import gzip
```

[95]:
```
# https://www.dataquest.io/blog/web-scraping-tutorial-python/
def getLinks(url): # scrape the url
    html_page = requests.get(url) #urllib2.urlopen(url)
    soup = BeautifulSoup(html_page.content, 'lxml')

    html = list(soup.children)[4] # gets the html tag
    nav = list(html.children)[5] # nav is the 'navigable string'

    # get all the links on the page
    links = []
    for link in nav.findAll('a', attrs={'href': re.compile("https://")}):
        links.append(link.get('href'))
    #print(len(links), 'links')

    # now get only the link we want to download
    downloads_list = []
    for item in links:
        if item[0:44] == 'https://cdn.rebrickable.com/media/downloads/' and␣
↪item[-3:]!='zip':
            print(item)
            downloads_list.append(item)
    print("We have", len(downloads_list), "links to download.")
    return downloads_list
```

[96]:
```
URL='https://rebrickable.com/downloads/'
linklist = getLinks(URL)
```

```
https://cdn.rebrickable.com/media/downloads/themes.csv.gz?1602930200.2377865
https://cdn.rebrickable.com/media/downloads/colors.csv.gz?1602930200.3457866
https://cdn.rebrickable.com/media/downloads/part_categories.csv.gz?1602930200.45
77868
https://cdn.rebrickable.com/media/downloads/parts.csv.gz?1602930200.8817878
https://cdn.rebrickable.com/media/downloads/part_relationships.csv.gz?1602930205
.2857969
https://cdn.rebrickable.com/media/downloads/elements.csv.gz?1602930201.1537883
https://cdn.rebrickable.com/media/downloads/sets.csv.gz?1602930201.6497893
https://cdn.rebrickable.com/media/downloads/minifigs.csv.gz?1602930201.8897898
https://cdn.rebrickable.com/media/downloads/inventories.csv.gz?1602930201.401789
https://cdn.rebrickable.com/media/downloads/inventory_parts.csv.gz?1602930204.62
17954
https://cdn.rebrickable.com/media/downloads/inventory_sets.csv.gz?1602930204.821
796
https://cdn.rebrickable.com/media/downloads/inventory_minifigs.csv.gz?1602930205
```

```
.0737965
We have 12 links to download.
```

Next download these links

[97]:
```
# Test the file_url construction
r = re.compile('[?]') # https://stackoverflow.com/a/2175096/8971265
file_url = r.split("https://cdn.rebrickable.com/media/downloads/themes.csv.gz?
 ↪1592816880.2813647")
file_url
```

[97]:
```
['https://cdn.rebrickable.com/media/downloads/themes.csv.gz',
 '1592816880.2813647']
```

[98]:
```
# Test the file name construction
r = re.compile('https://cdn.rebrickable.com/media/downloads/')
file_name = r.split(file_url[0])[1]
file_name
```

[98]:
```
'themes.csv.gz'
```

[99]:
```
def download_file(url, _path):

    """function to download a file at a given url and to a given path"""

    # get the file url w/o the slug
    r = re.compile('[?]') # https://stackoverflow.com/a/2175096/8971265
    file_url = r.split(url)
    print("file_url:", file_url[0])

    # get the file name
    r = re.compile('https://cdn.rebrickable.com/media/downloads/')
    file_name = r.split(file_url[0])[1]
    print("file_name:", file_name)

    # define headers dictionary containing the user agent (https://
 ↪stackoverflow.com/a/10606260/8971265)
    # https://www.scrapehero.com/
 ↪how-to-fake-and-rotate-user-agents-using-python-3/
    # https://developers.whatismybrowser.com/useragents/explore/
 ↪operating_system_name/windows/
    headers = {'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64)␣
 ↪AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36"}

    # download the file (https://stackoverflow.com/a/39217788/8971265)
    print("Downloading:", file_name)
    with requests.get(url, stream=True, headers=headers) as r:
        with open(os.path.join(_path,file_name), 'wb') as f:
```

```
            shutil.copyfileobj(r.raw, f)

    # unzip the file (https://stackoverflow.com/a/44712152/8971265)
    print("Unzipping:", file_name, "\n")
    with gzip.open(os.path.join(_path, file_name), 'rb') as f_src:
        with open(os.path.join(_path,file_name[:-3]), 'wb') as f_dst:
            shutil.copyfileobj(f_src, f_dst)

    return
```

[100]:
```
def fetch_files(_linklist,_path):

    """a function that creates a folder and download files to
    that folder from a list of links """

    if not os.path.isdir(_path): # if it is not there create it
        os.makedirs(_path)

    for link in linklist:

        # use the download_file function to download from the given link
        download_file(link, _path)
```

[101]:
```
# pass the folder name we want to create and save downloads in
data_path = os.path.join("data")

# download the files to that folder
fetch_files(linklist, data_path)
```

```
file_url: https://cdn.rebrickable.com/media/downloads/themes.csv.gz
file_name: themes.csv.gz
Downloading: themes.csv.gz
Unzipping: themes.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/colors.csv.gz
file_name: colors.csv.gz
Downloading: colors.csv.gz
Unzipping: colors.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/part_categories.csv.gz
file_name: part_categories.csv.gz
Downloading: part_categories.csv.gz
Unzipping: part_categories.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/parts.csv.gz
file_name: parts.csv.gz
Downloading: parts.csv.gz
```

```
Unzipping: parts.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/part_relationships.csv.gz
file_name: part_relationships.csv.gz
Downloading: part_relationships.csv.gz
Unzipping: part_relationships.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/elements.csv.gz
file_name: elements.csv.gz
Downloading: elements.csv.gz
Unzipping: elements.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/sets.csv.gz
file_name: sets.csv.gz
Downloading: sets.csv.gz
Unzipping: sets.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/minifigs.csv.gz
file_name: minifigs.csv.gz
Downloading: minifigs.csv.gz
Unzipping: minifigs.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/inventories.csv.gz
file_name: inventories.csv.gz
Downloading: inventories.csv.gz
Unzipping: inventories.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/inventory_parts.csv.gz
file_name: inventory_parts.csv.gz
Downloading: inventory_parts.csv.gz
Unzipping: inventory_parts.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/inventory_sets.csv.gz
file_name: inventory_sets.csv.gz
Downloading: inventory_sets.csv.gz
Unzipping: inventory_sets.csv.gz

file_url: https://cdn.rebrickable.com/media/downloads/inventory_minifigs.csv.gz
file_name: inventory_minifigs.csv.gz
Downloading: inventory_minifigs.csv.gz
Unzipping: inventory_minifigs.csv.gz
```

1.1 2. Load the data to a data dictionary

Function to load data.

[102]:
```
import pandas as pd
def load_csv_data(file_name, _path):
    file_path = os.path.join(_path, file_name)
    return pd.read_csv(file_path)
```

Make a list of files in the directory.

[103]:
```
files = os.listdir(data_path)
```

Make a list of csv files only.

[104]:
```
files_csv = [f for f in files if f[-4:] == '.csv'] # find all files ending with␣
↪'.csv'
display(files_csv)
```

```
['colors.csv',
 'df_sets_new.csv',
 'df_sets_used.csv',
 'elements.csv',
 'inventories.csv',
 'inventory_minifigs.csv',
 'inventory_parts.csv',
 'inventory_sets.csv',
 'minifigs.csv',
 'parts.csv',
 'part_categories.csv',
 'part_relationships.csv',
 'sets.csv',
 'themes.csv']
```

Make a data dictionary with the csv file names as keys.

[105]:
```
lego={} #https://stackoverflow.com/a/56217834/8971265
for i in range(len(files_csv)):
    lego[files_csv[i][:-4]] = load_csv_data(files_csv[i], data_path)

# print out one dataframe from the data dictionary to check
lego['themes'].head()
```

[105]:

	id	name	parent_id
0	1	Technic	NaN
1	2	Arctic Technic	1.0
2	3	Competition	1.0
3	4	Expert Builder	1.0
4	5	Model	1.0

1.2 3. Explore the data

(Taking some initial inspiration from this Kaggle notebook: Lego - let's play)

Colors

[106]:
```
df_colors = lego['colors']#.reset_index(drop=True)
df_colors.head()
```

[106]:

	id	name	rgb	is_trans
0	-1	[Unknown]	0033B2	f
1	0	Black	05131D	f
2	1	Blue	0055BF	f
3	2	Green	237841	f
4	3	Dark Turquoise	008F9B	f

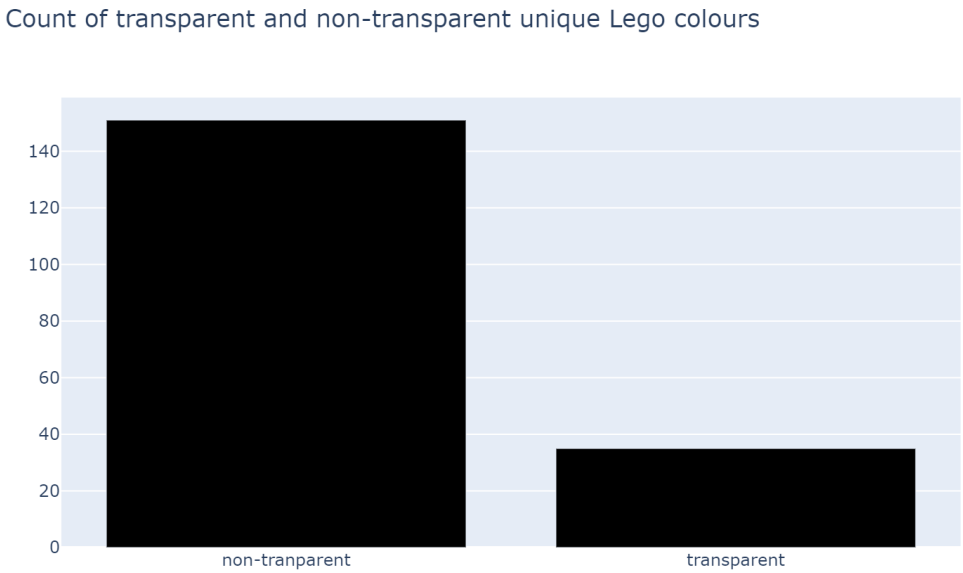
[147]:
```
# import plotly's 'graph objects' library
import plotly.graph_objs as go

# group dataframe by count of 'is_trans' column
group = df_colors.groupby('is_trans')[['is_trans']].count()
group.head()
```

[147]:

is_trans	is_trans
f	151
t	35

[154]:
```
# plot it
plot_name="Count of transparent and non-transparent unique Lego colours"
data = [go.Bar(x=['non-tranparent','transparent'],
               y=group['is_trans'],
               marker=dict(color='black'))
]
layout = go.Layout(title=plot_name)
fig = go.Figure(data=data, layout=layout)
fig.write_image("1.png", width=800, scale=2)
#fig.show()
```



Sets

[156]:
```
# dataframe of every lego set in the reirckable database
df_sets = lego['sets']
df_sets.head()
```

[156]:

	set_num	name	year	theme_id	num_parts
0	001-1	Gears	1965	1	43
1	0011-2	Town Mini-Figures	1978	84	12
2	0011-3	Castle 2 for 1 Bonus Offer	1987	199	0
3	0012-1	Space Mini-Figures	1979	143	12
4	0013-1	Space Mini-Figures	1979	143	12

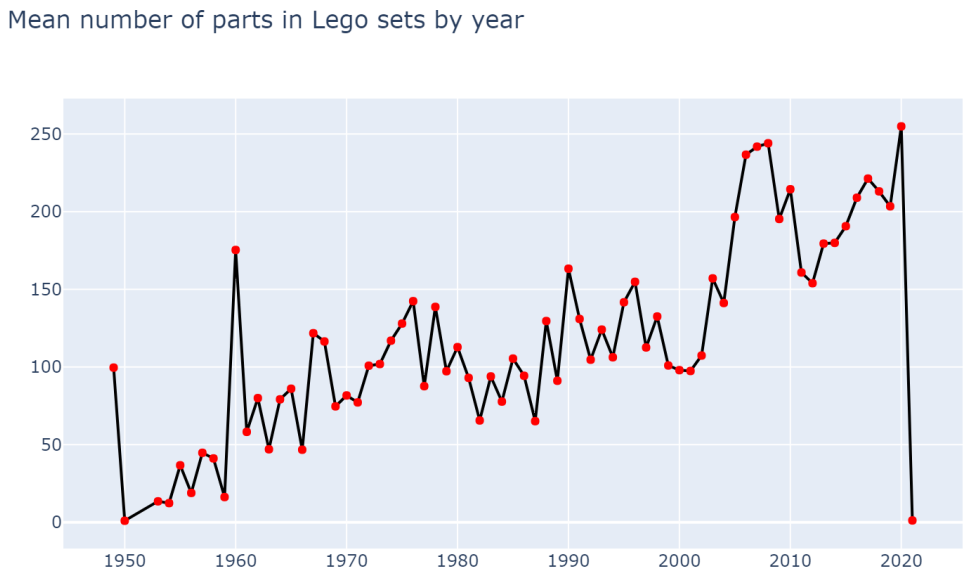
[157]:
```
# the years from that dataframe
df_sets.groupby(['year']).groups.keys()
```

[157]:
```
dict_keys([1949, 1950, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961,
1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974,
1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987,
1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013,
2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021])
```

[158]:
```
column_to_plot = 'num_parts'
# group by the 'column_to_plot' and display
group = df_sets.groupby('year')[[column_to_plot]].mean()
display(group.head())
```

```
      num_parts
year
1949  99.600000
1950   1.000000
1953  13.500000
1954  12.357143
1955  36.714286
```

[160]:
```
# plot it
data = [go.Scatter(x=group.index,
                   y=group[column_to_plot],
                   mode='lines+markers',
                   marker=dict(color='red'),
                   line=dict(color='black')
                  )
]
layout = go.Layout(title='Mean number of parts in Lego sets by year')
fig = go.Figure(data=data, layout=layout)
fig.write_image("2.png", width=800, scale=2)
```



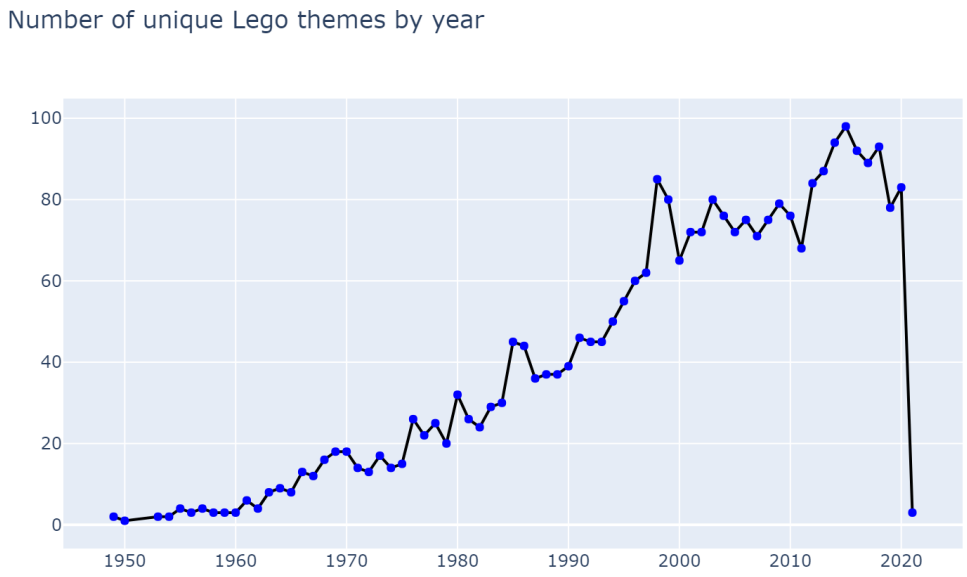
Themes

Group by number of unique theme_id ref

[161]:
```
group = df_sets.groupby('year')[['theme_id']].nunique()
display(group.head())
```

```
      theme_id
year
1949         2
1950         1
1953         2
1954         2
1955         4
```

[162]:
```
# plot it
data = [go.Scatter(x=group.index,
                   y=group['theme_id'],
                   mode='lines+markers',
                   marker=dict(color='blue'),
                   line=dict(color='black')
    )
]
layout = go.Layout(title='Number of unique Lego themes by year')
fig = go.Figure(data=data, layout=layout)
#fig.show()
fig.write_image("3.png", width=800, scale=2)
```



Back to the Colors

[164]:
```
df_colors.head()
```

[164]:
```
    id            name     rgb is_trans
0   -1       [Unknown]  0033B2        f
1    0           Black  05131D        f
2    1            Blue  0055BF        f
3    2           Green  237841        f
4    3  Dark Turquoise  008F9B        f
```

What's the difference between Inventories and Sets?

[165]:
```
df_inventories = lego['inventories']
df_inventories.head()
```

[165]:
```
    id  version set_num
0    1        1  7922-1
1    3        1  3931-1
2    4        1  6942-1
3   15        1  5158-1
4   16        1   903-1
```

[166]:
```
df_inventories.shape
```

[166]:
```
(26759, 3)
```

The inventories table has 'versions' of unique sets.

Look at the sets table again.

[167]:
```
df_sets.head()
```

[167]:
```
  set_num                         name  year  theme_id  num_parts
0   001-1                        Gears  1965         1         43
1  0011-2           Town Mini-Figures  1978        84         12
2  0011-3  Castle 2 for 1 Bonus Offer  1987       199          0
3  0012-1          Space Mini-Figures  1979       143         12
4  0013-1          Space Mini-Figures  1979       143         12
```

[168]:
```
# join the sets and inventories tables on 'set_num'
df_inv_sets = pd.merge(df_inventories, df_sets, on='set_num', how='outer').
 ↪sort_values(by='set_num')
df_inv_sets
```

[168]:
```
           id  version       set_num                                name  \
1925     2836        1          00-6                       Special Offer
12129   24696        1         001-1                               Gears
3423     5087        1        0011-2                  Town Mini-Figures
1492     2216        1        0011-3           Castle 2 for 1 Bonus Offer
955      1414        1        0012-1                  Space Mini-Figures
…        …           …            …                                   …
```

```
1303    1936        1  tominifigs-1            Town Minifig Packs 2-Pack
11031  16524        1    trucapam-1               Captain America Mosaic
9884   14717        1      tsuper-1                    Technic Super Set
3278    4868        1       vwkit-1                       Volkswagen Kit
507      758        1       wwgp1-1  Wild West Limited Edition Gift Pack

          year  theme_id  num_parts
1925    1985.0      67.0        0.0
12129   1965.0       1.0       43.0
3423    1978.0      84.0       12.0
1492    1987.0     199.0        0.0
955     1979.0     143.0       12.0
…          …         …          …
1303    2000.0      50.0        0.0
11031   2016.0     696.0       72.0
9884    1991.0      12.0        0.0
3278    1959.0     366.0       22.0
507     1996.0     476.0        0.0

[26759 rows x 7 columns]
```

[169]:
```
# take a look at the NaNs
df_inv_sets[df_inv_sets.isnull().any(axis=1)]
```

[169]:
```
           id  version     set_num name  year  theme_id  num_parts
15621   48649        1  fig-000001  NaN   NaN       NaN        NaN
15622   48650        1  fig-000002  NaN   NaN       NaN        NaN
15624   48681        1  fig-000003  NaN   NaN       NaN        NaN
15626   48758        1  fig-000004  NaN   NaN       NaN        NaN
15636   49548        1  fig-000005  NaN   NaN       NaN        NaN
…         …        …           …  …    …        …          …
26725   76081        1  fig-010711  NaN   NaN       NaN        NaN
26726   76082        1  fig-010712  NaN   NaN       NaN        NaN
26728   76167        1  fig-010713  NaN   NaN       NaN        NaN
26729   76168        1  fig-010714  NaN   NaN       NaN        NaN
26739   76650        1  fig-010715  NaN   NaN       NaN        NaN

[10470 rows x 7 columns]
```

The difference between the number of unique inventory ids and set_num's is that fact that figures don't have a set number.

[170]:
```
# drop NaNs
df_inv_sets.dropna(axis=0, inplace=True)
df_inv_sets
```

[170]:
```
          id  version       set_num                                  name  \
1925    2836        1          00-6                         Special Offer
12129  24696        1         001-1                                 Gears
3423    5087        1        0011-2                   Town Mini-Figures
1492    2216        1        0011-3          Castle 2 for 1 Bonus Offer
955     1414        1        0012-1                   Space Mini-Figures
…       …       …           …                                        …
1303    1936        1  tominifigs-1           Town Minifig Packs 2-Pack
11031  16524        1    trucapam-1              Captain America Mosaic
9884   14717        1      tsuper-1                   Technic Super Set
3278    4868        1       vwkit-1                      Volkswagen Kit
507      758        1       wwgp1-1  Wild West Limited Edition Gift Pack

          year  theme_id  num_parts
1925    1985.0      67.0        0.0
12129   1965.0       1.0       43.0
3423    1978.0      84.0       12.0
1492    1987.0     199.0        0.0
955     1979.0     143.0       12.0
…        …        …          …
1303    2000.0      50.0        0.0
11031   2016.0     696.0       72.0
9884    1991.0      12.0        0.0
3278    1959.0     366.0       22.0
507     1996.0     476.0        0.0

[16289 rows x 7 columns]
```

[171]:
```
# join the result with 'inventory_parts' table to get a table of all part for␣
↪all sets
df_inventory_parts = lego['inventory_parts']
df_all = pd.merge(df_inv_sets, df_inventory_parts, how='outer', left_on='id',␣
↪right_on='inventory_id')
df_all
```

[171]:
```
              id  version set_num           name    year  theme_id  num_parts  \
0         2836.0      1.0    00-6  Special Offer  1985.0      67.0        0.0
1        24696.0      1.0   001-1          Gears  1965.0       1.0       43.0
2        24696.0      1.0   001-1          Gears  1965.0       1.0       43.0
3        24696.0      1.0   001-1          Gears  1965.0       1.0       43.0
4        24696.0      1.0   001-1          Gears  1965.0       1.0       43.0
…            …      …      …              …      …        …         …
853171       NaN      NaN     NaN            NaN     NaN       NaN        NaN
853172       NaN      NaN     NaN            NaN     NaN       NaN        NaN
853173       NaN      NaN     NaN            NaN     NaN       NaN        NaN
853174       NaN      NaN     NaN            NaN     NaN       NaN        NaN
853175       NaN      NaN     NaN            NaN     NaN       NaN        NaN
```

```
        inventory_id      part_num  color_id  quantity is_spare
0                NaN           NaN       NaN       NaN      NaN
1            24696.0          132a       7.0       4.0        f
2            24696.0          3020      15.0       4.0        f
3            24696.0         3062c      15.0       1.0        f
4            24696.0      3404bc01      15.0       4.0        f
…                …             …         …         …        …
853171       76650.0          2446     484.0       1.0        f
853172       76650.0          2447      40.0       1.0        f
853173       76650.0   3626cpr3026      14.0       1.0        f
853174       76650.0        970c00       0.0       1.0        f
853175       76650.0  973pr5038c01     288.0       1.0        f

[853176 rows x 12 columns]
```

[172]:
```
# drop NaNs
df_all.dropna(axis=0, inplace=True)
```

[173]:
```
# join with colors table to bring in the piece-wise color codes
df_all = pd.merge(df_all, df_colors, left_on='color_id', right_on='id',␣
 ↪how='outer')
df_all.head()
```

[173]:
```
      id_x  version set_num               name_x    year  theme_id  num_parts  \
0  24696.0      1.0   001-1                Gears  1965.0       1.0       43.0
1  24696.0      1.0   001-1                Gears  1965.0       1.0       43.0
2  13351.0      1.0  0016-1  Castle Mini Figures  1978.0     186.0       15.0
3  24702.0      1.0   003-1  Master Mechanic Set  1966.0     366.0      403.0
4  24702.0      1.0   003-1  Master Mechanic Set  1966.0     366.0      403.0

   inventory_id part_num  color_id  quantity is_spare  id_y      name_y  \
0       24696.0     132a       7.0       4.0        f     7  Light Gray
1       24696.0       36       7.0       4.0        f     7  Light Gray
2       13351.0    3847a       7.0       3.0        f     7  Light Gray
3       24702.0     132a       7.0       8.0        f     7  Light Gray
4       24702.0       36       7.0       4.0        f     7  Light Gray

      rgb is_trans
0  9BA19D        f
1  9BA19D        f
2  9BA19D        f
3  9BA19D        f
4  9BA19D        f
```

[174]:
```
# drop NaNs and create a new table with the columns we are interested in
df_all.dropna(axis=0, inplace=True)
```

```
df_all_colors =␣
 ↪df_all[['set_num','name_x','year','theme_id','color_id','quantity','name_y','rgb','is_trans
df_all_colors
```

[174]:

	set_num	name_x	year	theme_id	color_id	quantity	name_y	rgb	is_trans
0	001-1	Gears	1965.0	1.0	7.0	4.0	Light Gray	9BA19D	f
1	001-1	Gears	1965.0	1.0	7.0	4.0	Light Gray	9BA19D	f
2	0016-1	Castle Mini Figures	1978.0	186.0	7.0	3.0	Light Gray	9BA19D	f
3	003-1	Master Mechanic Set	1966.0	366.0	7.0	8.0	Light Gray	9BA19D	f
4	003-1	Master Mechanic Set	1966.0	366.0	7.0	4.0	Light Gray	9BA19D	f
…	…	…	…	…	…	…	…	…	…
803683	LEGO-Modulex-1	Unused Modulex parts sold by LEGO	2019.0	408.0	1037.0	1.0	Modulex Violet	BD7D85	f
803684	LEGO-Modulex-1	Unused Modulex parts sold by LEGO	2019.0	408.0	1037.0	1.0	Modulex Violet	BD7D85	f
803685	LEGO-Modulex-1	Unused Modulex parts sold by LEGO	2019.0	408.0	1024.0	1.0	Modulex Pink Red	F45C40	f
803686	LEGO-Modulex-1	Unused Modulex parts sold by LEGO	2019.0	408.0	1035.0	1.0	Modulex Medium Blue	61AFFF	f
803687	LEGO-Modulex-1	Unused Modulex parts sold by LEGO	2019.0	408.0	1027.0	1.0	Modulex Light Yellow	FFE371	f

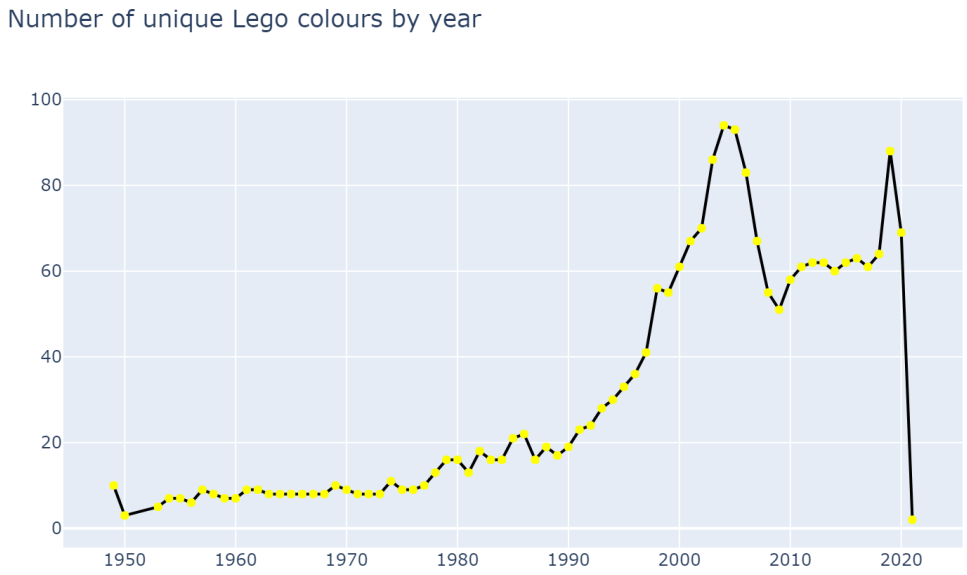
[803688 rows x 9 columns]

[175]:
```
# group by year and aggregate the by number of unique colour codes
group = df_all_colors.groupby('year')[['rgb']].nunique()
display(group.head())
```

year	rgb
1949.0	10
1950.0	3
1953.0	5
1954.0	7
1955.0	7

[179]:
```
# plot it
data = [go.Scatter(x=group.index,
```

```
                    y=group['rgb'],
                    mode='lines+markers',
                    marker=dict(color='yellow'),
                    line=dict(color='black')
    )
]
layout = go.Layout(title='Number of unique Lego colours by year')
fig = go.Figure(data=data, layout=layout)
fig.write_image("4.png", width=800, scale=2)
```



[180]:
```
# get the list of unique colours for each year
colors = list(df_all_colors['rgb'].unique())
years = group.index.astype(int)
years
```

[180]:
```
Int64Index([1949, 1950, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961,
            1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972,
            1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983,
            1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994,
            1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005,
            2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016,
            2017, 2018, 2019, 2020, 2021],
           dtype='int64')
```

[181]:
```
# get the unique color lists by year
colors_list_by_year = []
```

```
for year in years:
    temp = df_all_colors[df_all_colors['year']==year]
    colors_for_year = list(temp['rgb'].unique())
    colors_list_by_year.append(colors_for_year)
```

[182]:
```
colors_list_by_year[0:9]
```

[182]:
```
[['FFFFFF',
  'C91A09',
  'F2CD37',
  '0055BF',
  '237841',
  '4B9F4A',
  'CA1F08',
  '1E601E',
  'F3C305',
  '039CBD'],
 ['C91A09', '0055BF', '237841'],
 ['FFFFFF', 'C91A09', 'F2CD37', '0055BF', '237841'],
 ['9BA19D', 'FFFFFF', 'C91A09', 'F2CD37', '0055BF', 'FCFCFC', '237841'],
 ['FFFFFF', 'C91A09', 'F2CD37', '0055BF', 'FCFCFC', '237841', '05131D'],
 ['FFFFFF', 'C91A09', 'F2CD37', '0055BF', 'FCFCFC', '05131D'],
 ['9BA19D',
  'FFFFFF',
  'C91A09',
  'F2CD37',
  '0055BF',
  '05131D',
  'FCFCFC',
  '237841',
  'A5A9B4'],
 ['FFFFFF',
  'C91A09',
  'F2CD37',
  '0055BF',
  '05131D',
  'FCFCFC',
  '237841',
  'A5A9B4'],
 ['9BA19D', 'FFFFFF', 'C91A09', 'F2CD37', '0055BF', 'FCFCFC', '05131D']]
```

[183]:
```
# define a function to count features
def count_a_feature(feature, year, _df):

    """function to count values of a given feature
    and its proportion of the total count for a
    dataframe passed, and return a dataframe
```

```
    with the result for a given year"""

    temp = _df[_df['year']==year] # filter for the year
    col1 = temp[feature].value_counts()
    total = temp[feature].value_counts().sum()
    col2 = round(100*temp[feature].value_counts() / total, 1)
    data = {'count': col1, 'pc': col2}
    temp = pd.DataFrame(data).sort_values(by='count', ascending=False)
    #display(temp)

    return temp, feature
```

[184]:
```
# test it
count_a_feature('rgb', 1950, df_all_colors)
```

[184]:
```
(        count    pc
 0055BF     10  45.5
 237841      9  40.9
 C91A09      3  13.6,
 'rgb')
```

Create a dataframe of all unique colours (index) by year (columns) with the proportion of those colours for each year as the values.

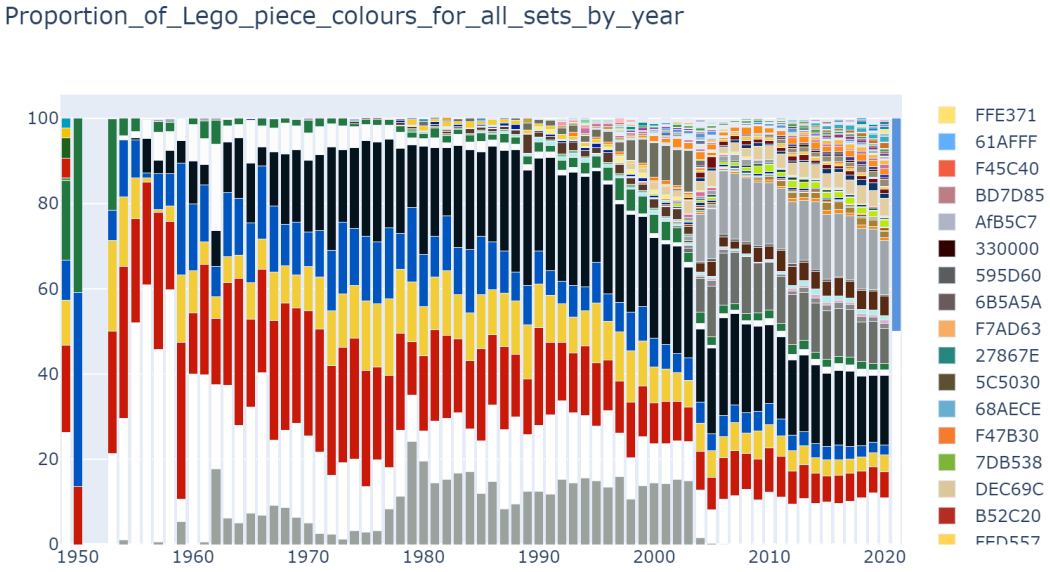
[185]:
```
import numpy as np
# get the list of unique colours for each year
colors = list(df_all_colors['rgb'].unique())
# get the unique years
year = list(df_all_colors['year'].unique())
# set up a new dataframe
df_stack = pd.DataFrame(index=colors, columns=years)
# get the portion of all colors for each colour in each year
for year in years:
    df_feature, feature = count_a_feature('rgb', year, df_all_colors)
    for color in colors:
        try:
            df_stack.loc[color, year] = df_feature.loc[color, 'pc']
        except:
            df_stack.loc[color, year] = np.nan
df_stack = df_stack[::-1] # [::-1] reverses order
# df_stack
```

[191]:
```
# function to plot it
def stacked_plot(plot_name,_df):

    data = []
    font_size=12
```

```
    for color in colors:
        trace = go.Bar(x=_df.columns,
                       y=_df.loc[str(color)],
                       name=str(color),
                       marker=dict(color='#'+str(color))
        )
        data.append(trace)
    layout = go.Layout(title=plot_name,
                       barmode='stack',
                       font=dict(size=font_size))
    fig = go.Figure(data=data, layout=layout)
    #fig.show()
    fig.write_image(f"{plot_name}.png", width=800, scale=2)
```

[194]:
```
stacked_plot("Proportion_of_Lego_piece_colours_for_all_sets_by_year", df_stack)
```

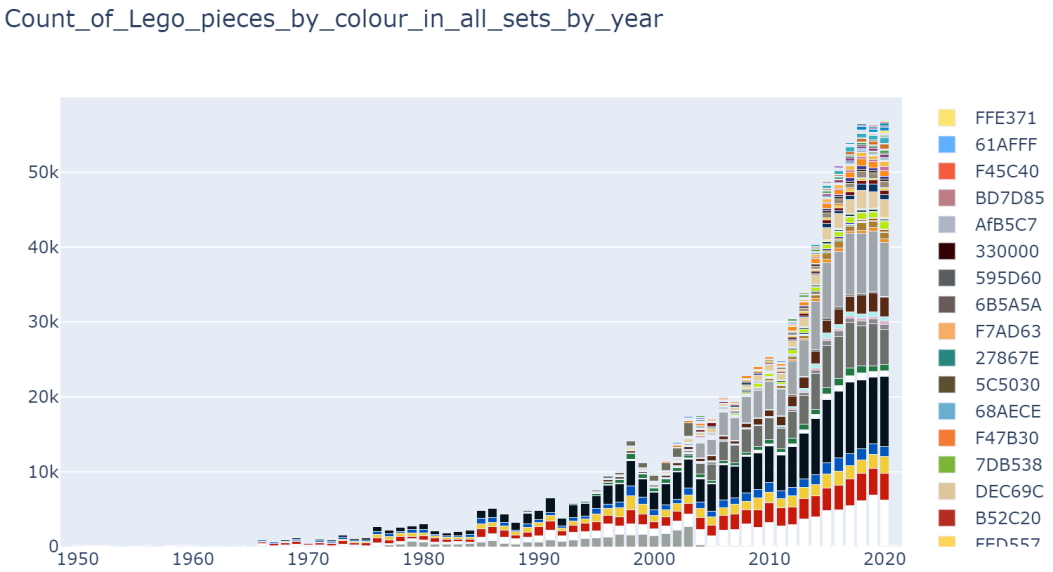


Now plot by count.

[195]:
```
import numpy as np
# get the list of unique colours for each year
colors = list(df_all_colors['rgb'].unique())
# get the unique years
year = list(df_all_colors['year'].unique())
# set up a new dataframe
df_stack_count = pd.DataFrame(index=colors, columns=years)
# get the portion of all colors for each colours in each year
```

```
for year in years:
    df_feature, feature = count_a_feature('rgb', year, df_all_colors)
    for color in colors:
        try:
            df_stack_count.loc[color, year] = df_feature.loc[color, 'count']
        except:
            df_stack_count.loc[color, year] = np.nan
df_stack_count = df_stack_count[::-1] # reverse order
#df_stack_count
```

[196]:
```
stacked_plot("Count_of_Lego_pieces_by_colour_in_all_sets_by_year",␣
 ↪df_stack_count)
# could filter this by theme
# could add title to legend as 'rbg'
```



1.3 4. Get market price data

Use the Brick Link API to get guide price data of all sets

[ ]:
```
from bricklink_api.auth import oauth
from bricklink_api.catalog_item import get_price_guide, Type, NewOrUsed

# authorisation details from a seller account https://www.bricklink.com/v2/api/
 ↪register_consumer.page
consumer_key = "9667E5445FA043538C4C34B2AFE74CFB"
consumer_secret = "D9AAF83E1F574360896C81A3072B48C6"
token_value = "8AFE17DBFFAB46629A6214E5EAD57F37"
```

```
token_secret = "967AB439A70E4D45831686F1E200817C"
auth = oauth(consumer_key, consumer_secret, token_value, token_secret)

# Example, get price guide for a used 42100-1 (Lego Technic Liebherr R 9800)
json_obj = get_price_guide(Type.SET, "42100-1", new_or_used='U', auth=auth)
#json_obj
```

[ ]:
```
def get_set_price_data(_set, N_or_U):

    """function to get the price data of a given
    set number as new (N) or used (U)"""

    try:
        json_obj = get_price_guide(Type.SET, _set, new_or_used=N_or_U,␣
 ↪auth=auth)
        average = json_obj['data']['avg_price']
        minimum = json_obj['data']['min_price']
        maximum = json_obj['data']['max_price']
        quantity = json_obj['data']['total_quantity']
        currency = json_obj['data']['currency_code']
    except:
        average, minimum, maximum, quantity, currency = 0., 0., 0., 0., 'none'

    return average, minimum, maximum, quantity, currency
```

Get all new price data for all sets

[ ]:
```
#https://stackoverflow.com/a/20627316/8971265
pd.options.mode.chained_assignment = None  # default='warn'

import time

# new columns
df_sets['price_mean_N'] = ''
df_sets['price_min_N'] = ''
df_sets['price_max_N'] = ''
df_sets['set_qty_N'] = ''
df_sets['currency_N'] = ''

i = 0 # row counter
for _set in df_sets['set_num']:

    print('Set:', _set)

    # get the set price data
    result = get_set_price_data(_set, 'N')
    df_sets.loc[[i], ['price_mean_N']] = result[0]
```

```
    df_sets.loc[[i], ['price_min_N']] = result[1]
    df_sets.loc[[i], ['price_max_N']] = result[2]
    df_sets.loc[[i], ['set_qty_N']] = result[3]
    df_sets.loc[[i], ['currency_N']] = result[4]

    # print the row number and Bricklink result
    print('Row:', i, result)

    #bump the counter
    i += 1

    # sleep 24 hours every 4000 calls (to avoid exceeding the API limitations)
    if i % 4000 == 0:

        print("Saving...")
        print(f"Set, i={i}, day={time.localtime().tm_mday}")
        # wait 24 hours
        print("Sleeping...")
        time.sleep(60*60*24)

df_sets.to_csv("data/df_sets_new.csv")
del df_sets
```

Load from csv

[197]:
```
df_sets = pd.read_csv("data/df_sets_new.csv", index_col=0)
df_sets.tail()
```

[197]:

	set_num	name	year	theme_id	num_parts	price_mean_N	price_min_N	price_max_N	set_qty_N	currency_N
15492	wwgp1-1	Wild West Limited Edition Gift Pack	1996	476	0	0.0000	0.0000	0.0000	0.0	USD
15493	XMASTREE-1	Christmas Tree	2019	410	26	0.0000	0.0000	0.0000	0.0	none
15494	XWING-1	Mini X-Wing Fighter	2019	158	60	0.0000	0.0000	0.0000	0.0	none
15495	XWING-2	X-Wing Trench Run	2019	158	52	0.0000	0.0000	0.0000	0.0	none
15496	YODACHRON-1	Yoda Chronicles Promotional Set	2013	158	413	265.2236	265.2236	265.2236	2.0	USD

[198]:
```
# Typical example, The Chicken Coop
df_sets[df_sets['set_num']=='21140-1']
```

[198]:

	set_num	name	year	theme_id	num_parts	price_mean_N	\
1902	21140-1	The Chicken Coop	2018	577	198	22.8214	

```
      price_min_N  price_max_N  set_qty_N currency_N
1902        14.99      35.5129      328.0        USD
```

[199]:
```
# remove the zero-priced sets
df_prices_N = df_sets[df_sets['price_mean_N'] > 0.0]
df_prices_N
```

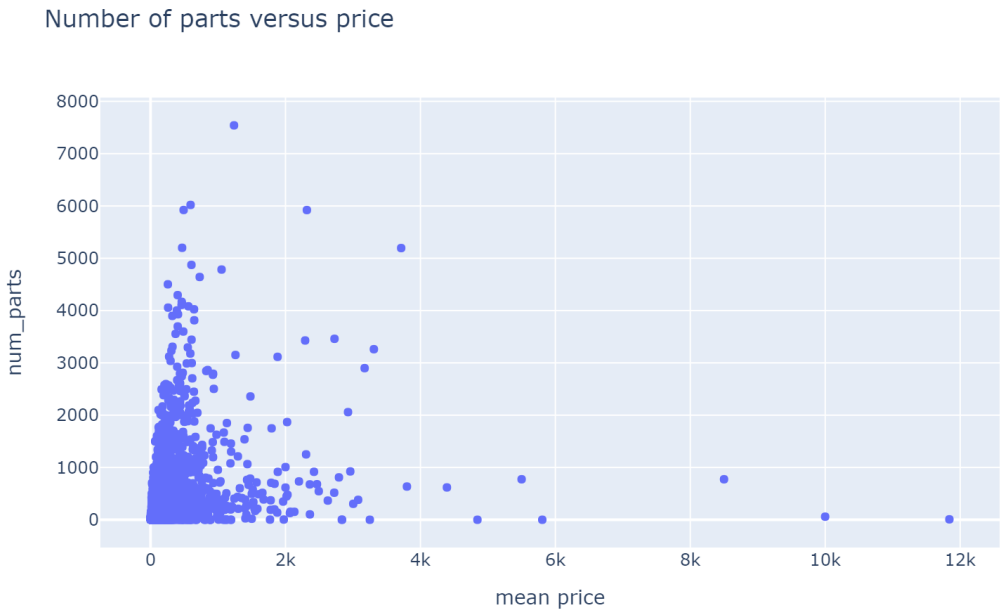
[199]:
```
           set_num                                                  name  year  \
30           044-1                                 Basic Building Set  1968
51           074-1                                     Pre-School Set  1976
67           088-1                                          Super Set  1969
68         10000-1                                       Guarded Inn  2001
70         10001-1                                        Metroliner  2001
…              …                                                  …     …
15481    Watford-1  LEGO Store Grand Opening Exclusive Set, Watfor…  2013
15482  Wauwatosa-1  LEGO Store Grand Opening Exclusive Set, Mayfai…  2012
15489  Wiesbaden-1  LEGO Store Grand Opening Exclusive Set, Wiesba…  2010
15490    WILLIAM-1                                          Will.i.am  2016
15496  YODACHRON-1                  Yoda Chronicles Promotional Set  2013

       theme_id  num_parts  price_mean_N  price_min_N  price_max_N  set_qty_N  \
30          366        225     165.6266     165.6266     165.6266        1.0
51          505         20      60.0000      60.0000      60.0000        1.0
67          469        615    2000.0000    2000.0000    2000.0000        1.0
68          199        256     262.5159     199.0000     352.2405       14.0
70          233        787    1477.5102     899.9900    3640.4192        5.0
…             …          …            …            …            …          …
15481       408         15     236.0687     236.0687     236.0687        1.0
15482       408         15     191.4508     150.0000     234.8883        5.0
15489       408        146     211.2815     211.2815     211.2815        1.0
15490       535          3    1770.4561    1770.4561    1770.4561        1.0
15496       158        413     265.2236     265.2236     265.2236        2.0

      currency_N
30           USD
51           USD
67           USD
68           USD
70           USD
…              …
15481        USD
15482        USD
15489        USD
15490        USD
15496        USD
```

```
[9527 rows x 10 columns]
```

[201]:
```
# plot scatter of price and num_parts
data = [go.Scatter(x=df_prices_N['price_mean_N'],
                   y=df_prices_N['num_parts'],
                   mode='markers'
    )
]
layout=go.Layout(title='Number of parts versus price',
                 xaxis=dict(title='mean price'),
                 yaxis=dict(title='num_parts'))
fig = go.Figure(data=data, layout=layout)
#fig.show()
fig.write_image("5.png", width=800, scale=2)
```

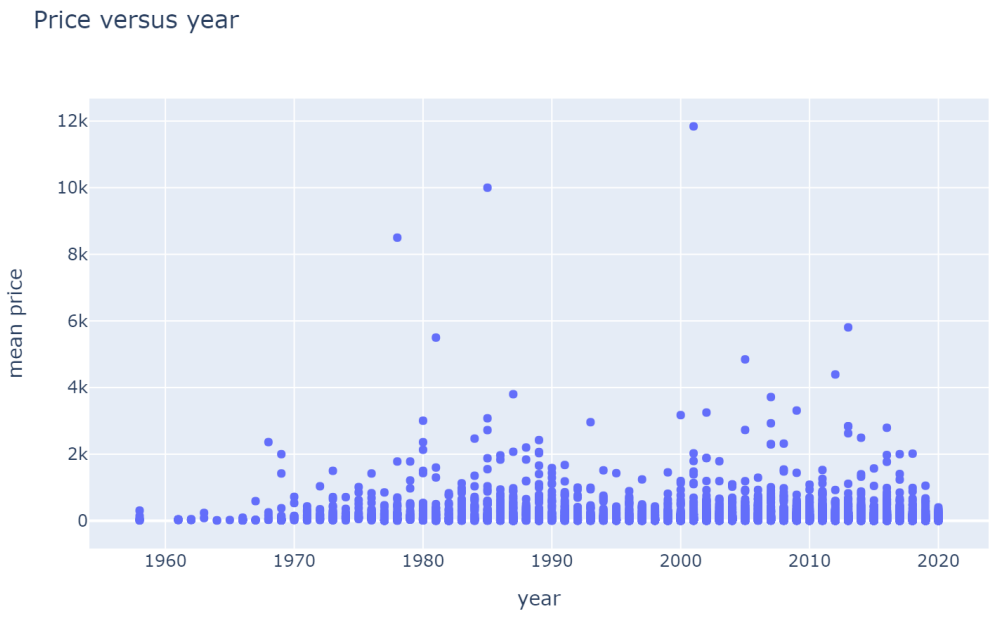


[202]:
```
corr_matrix = df_prices_N.corr()
corr_matrix['price_mean_N'].sort_values(ascending=False)
```

[202]:
```
price_mean_N    1.000000
price_min_N     0.972301
price_max_N     0.896906
num_parts       0.304013
set_qty_N      -0.077959
theme_id       -0.084471
year           -0.141753
Name: price_mean_N, dtype: float64
```

This result tells us that the miniumum and maximum prices are strinly correlated to the mean price, both having Pearson correlation number close to 1. But that information is not useful. The number of parts in a set is correlated to the mean price having a score of 0.3. The positive score means that the higher the number of parts, the higher the price. Interestingly, we can see that the year has a slight negative correlation to price; the lower the year the higher the price.

[203]:
```
# plot scatter of price and num_parts
data = [go.Scatter(y=df_prices_N['price_mean_N'],
                   x=df_prices_N['year'],
                   mode='markers',
                   hovertext=df_prices_N['name']
    )
]
layout=go.Layout(title='Price versus year',
                   yaxis=dict(title='mean price'),
                   xaxis=dict(title='year'))
fig = go.Figure(data=data, layout=layout)
#fig.show()
fig.write_image("6.png", width=800, scale=2)
```



[145]:
```
# (perhaps identify and plot by theme spearately)
# list of unique themes names
df_themes.name.unique()
```

[145]:
```
array(['Arctic Technic', 'Competition', 'Expert Builder', 'Model',
       'Airport', 'Construction', 'Farm', 'Fire', 'Harbor', 'Off-Road',
```

'Race', 'Riding Cycle', 'Robot', 'Traffic', 'RoboRiders',
'Speed Slammers', 'Star Wars', 'Supplemental', 'Throwbot Slizer',
'Universal Building Set', 'Basic Model', 'Castle', 'Train',
'Creature', 'Food & Drink', 'Building', 'Cargo', 'Basic Set',
'Recreation', 'Mecha', 'Arctic', 'City', 'Coast Guard', 'Hospital',
'Police', 'Trains', 'Classic Town', 'Station', 'Post Office',
'Divers', 'Extreme Team', 'Launch Command', 'Outback', 'Paradisa',
'Res-Q', 'Space Port', 'Town Jr.', 'Gas Station', 'Town Plan',
'World City', 'Drome Racers', 'Ferrari', 'Lamborghini',
'Power Racers', 'Radio Control', 'Speed Racer', 'Tiny Turbos',
'Track System', 'Williams F1', 'World Racers', 'Xalax',
'Alien Conquest', 'Blacktron I', 'Blacktron II', 'Classic Space',
'Exploriens', 'Futuron', 'Ice Planet 2002', 'Insectoids',
'Life On Mars', 'M:Tron', 'Mars Mission', 'RoboForce',
'Space Police I', 'Space Police II', 'Space Police III', 'Spyrius',
'UFO', 'Unitron', 'Galaxy Squad', 'Pirates I', 'Imperial Armada',
'Imperial Soldiers', 'Islanders', 'Pirates II', 'Pirates III',
'Mini', 'Bricktober', 'Ultimate Collector Series', 'Black Falcons',
'Black Knights', 'Classic Castle', 'Crusaders', 'Dark Forest',
'Fantasy Era', 'Forestmen', 'Fright Knights', 'Kingdoms',
'Knights Kingdom I', 'Knights Kingdom II', 'Lion Knights',
'My Own Creation', 'Royal Knights', 'Wolfpack', 'Advent',
'Belville', 'Classic Basic', 'Clikits', 'Creator', 'Pirates',
'Friends', 'Christmas', 'Easter', 'Halloween', 'Thanksgiving',
'Valentine', '12V', '4.5V', '9V', 'My Own Train', 'RC Train',
'Skylines', 'Technic', 'NXT', 'RCX', 'EV3', 'Jurassic Park III',
'Mosaic', 'Jack Stone', 'Spider-Man', 'Desert', 'Dino Island',
'Jungle', 'Orient Expedition', 'Ultra Agents', 'Mission Deep Sea',
'Mission Deep Freeze', 'Aquanauts', 'Aquaraiders I',
'Aquaraiders II', 'Aquasharks', 'Hydronauts', 'Stingrays',
'Fairy-Tale', 'Golden Land', 'Playhouse', 'Agori', 'Barraki',
'Battle Vehicles', 'Bohrok', 'Bohrok Va', 'Bohrok-Kal',
'Glatorian', 'Glatorian Legends', 'Matoran of Light',
'Matoran of Mahri Nui', 'Matoran of Mata Nui',
'Matoran of Metru Nui', 'Matoran of Voya Nui', 'Mistika',
'Phantoka', 'Piraka', 'Playsets', 'Rahaga', 'Rahi', 'Rahkshi',
'Stars', 'Titans', 'Toa', 'Toa Hagah', 'Toa Hordika', 'Toa Inika',
'Toa Mahri', 'Toa Metru', 'Toa Nuva', 'Tohunga', 'Turaga', 'Vahki',
'Visorak', 'Warriors', 'Protectors', 'Skull Spiders', 'Toa Okoto',
'HO 1:87 Vehicles', 'Jumbo Bricks', 'Vehicle', 'Wooden Box Set',
'Heroes', 'Vehicles', 'Villains', 'Monthly Mini Model Build',
'Pick A Model', 'Space', 'Western', 'Bionicle', 'Holiday',
'Airjitzu', 'Adventurers', 'Aquazone', 'Fabuland', 'Primo',
'Scala', 'Town', 'Basketball', 'Gravity Games', 'Hockey', 'Soccer',
'Basic', 'Ferries', 'Gears', 'Cowboys', 'Indians', 'Game',
'Key Chain', 'Cars', 'Boat', 'Building Set with People', 'Classic',
'Dinosaurs', 'Duplo and Explore', 'Learning', 'Mindstorms', 'WeDo',

```
'Samsonite', 'Service Packs', 'Soft Bricks', 'Control Lab', 'eLAB',
'Series 1 Minifigures', 'Series 2 Minifigures',
'Series 3 Minifigures', 'Series 4 Minifigures',
'Series 5 Minifigures', 'Series 6 Minifigures',
'Series 7 Minifigures', 'Series 8 Minifigures',
'Series 9 Minifigures', 'Series 10 Minifigures', 'Team GB',
'Series 11 Minifigures', 'Series 12 Minifigures',
'The LEGO Movie Series', 'The Simpsons', 'Series 13 Minifigures',
'Series 14 Minifigures', 'The Simpsons Series 2',
'Series 15 Minifigures', 'Disney Series 1',
'Series 16 Minifigures', 'DFB Minifigures', 'The Hobbit',
'The Lord of the Rings', 'Speedorz', 'Constraction',
'Legend Beasts', 'Series 1', 'Series 2', 'Series 3', 'Series 4',
'Series 5', 'Series 6', 'Series 7', 'Series 8', 'Series 9',
'DC Comics Super Heroes', 'Disney Princess', 'Marvel Super Heroes',
'Ninjago', 'The LEGO Batman Movie', 'Series 17 Minifigures',
'Ninjago The Movie', 'Cities of Wonders',
'Jurassic World: Fallen Kingdom', 'Series 18 Minifigures',
'Bob the Builder', 'Disney Planes', 'Dora the Explorer',
'Miles From Tomorrowland', 'Spiderman', 'Thomas & Friends',
'Toy Story', 'Winnie the Pooh', 'My Town',
'Jake and the Never Land Pirates', 'Sofia the First',
"Disney's Mickey Mouse", 'Doc McStuffins', 'Legoville',
'DC Comics', 'Powered Up',
'Harry Potter and Fantastic Beasts Series 1', 'Action Wheelers',
'Dino', 'Dolls', 'Little Forest Friends', 'Little Robots',
'Princess Castle', 'Rattles', 'Toolo', 'Zooters',
'Fantastic Beasts', 'The LEGO Movie II', 'Creator 3-in-1',
'Creator Expert', 'Early Creator', 'The LEGO Movie Series II',
'Disney Series 2', 'Mars Exploration', 'Series 19 Minifigures',
'DC Super Heroes', 'Frozen II',
'Harry Potter and Fantastic Beasts Series 2',
'Series 20 Minifigures', 'Batman', 'UCS', 'Justice League',
'Superman', 'Avengers', 'Guardians of the Galaxy', 'Iron Man',
'X-Men', 'Harry Potter'], dtype=object)
```

[ ]:
```
#---------example of price guide-------
```

[91]:
```
# get price guide
json_obj = get_price_guide(Type.SET, "21138-1", new_or_used='U', auth=auth)
json_obj
```

[91]:
```
{'meta': {'description': 'OK', 'message': 'OK', 'code': 200},
 'data': {'item': {'no': '21138-1', 'type': 'SET'},
  'new_or_used': 'U',
  'currency_code': 'USD',
  'min_price': '19.0000',
```

```
'max_price': '29.2403',
'avg_price': '24.0801',
'qty_avg_price': '24.0801',
'unit_quantity': 3,
'total_quantity': 3,
'price_detail': [{'quantity': 1,
  'unit_price': '29.2403',
  'shipping_available': True,
  'qunatity': 1},
 {'quantity': 1,
  'unit_price': '24.0000',
  'shipping_available': False,
  'qunatity': 1},
 {'quantity': 1,
  'unit_price': '19.0000',
  'shipping_available': False,
  'qunatity': 1}]}}
```

[ ]: