In [1]:

```
import re
import string
import json
import os
import glob
import shutil
from pprint import pprint
from functools import partial

import joblib
import matplotlib.pyplot as plt
import nltk
import numpy as np
import scipy as sp
import pandas as pd
import seaborn as sns
import sacremoses
from sklearn.base import clone
from sklearn.compose import (
    ColumnTransformer,
    make_column_selector,
    make_column_transformer,
)
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.metrics import (
    accuracy_score,
    balanced_accuracy_score,
    classification_report,
)


# Set Seaborn theme and default palette
sns.set_context("talk")
sns.set_theme(font_scale=1.25, style="whitegrid")
sns.set_palette("deep", desat=0.85, color_codes=True)

# Turn on inline plotting
%matplotlib inline

# Load Black auto-formatter
%load_ext nb_black
```

In [2]:

```
# Import my modules
from tools import cleaning, plotting, language as lang, outliers, utils
from tools.sklearn import selection

RUN_LANGDETECT = False

# Run time-consuming grid searches
RUN_SWEEPS = frozenset()

FIT_MODELS = frozenset()

# Set my default MPL settings
plt.rcParams.update(plotting.MPL_DEFAULTS)

# Enable automatic reloading
%load_ext autoreload
%autoreload 2
```

Business Problem

Amazon has asked me to build a product classifier for two purposes: (1) integrating new products into their classification scheme, and (2) flagging products which are probably misclassified. They requested that I make some recommendations related to product classification and its uses.

Accuracy is my highest priority, but I have a taste for interpretability and transparency, so the classifier I develop is sure to yield some insights about the data.

Sourcing the Data

The Amazon product data I've chosen doesn't come directly from Amazon, but rather from three AI researchers, Jianmo Ni, Jiacheng Li, and Julian McAuley, who gathered it for their paper "Justifying Recommendations using Distantly-Labeled Reviews and Fine-grained Aspects." The review data extends from May 1996 to October 2018, which is about when they released the update. Their focus was primarily on reviews, but the dataset also has metadata for ~15-million products. The researchers don't say how they acquired the data, but judging from the HTML tags and chunks of JavaScript, they probably scraped it.

Since the full dataset is ~15M samples and ~13GB, I've created a separate notebook called `big_clean.ipynb` in which I download, reformat, and scrub the data until I'm left with ~4M samples. Despite the long and tortuous journey that is `big_clean.ipynb`, there's still much more to go.

Preparing the Data

I begin by loading the data and doing a preliminary cleaning check.

In [3]:
```
df = pd.read_parquet(
    "data/amazon_clean",
    engine="pyarrow",
    columns=[
        "text",
        "title",
        "brand",
        "main_cat",
    ],
)

df
```

Out[3]:

asin	text	title	brand	main_cat
1441072241	come tell me how you lived: native american hi...	come tell me how you lived: native american hi...	enter the arena	all beauty
6090113980	prayer rug carpet mat gebetsteppich islamic mu...	prayer rug carpet mat gebetsteppich islamic mu...	bonballoon	all beauty
7250468162	no7 stay perfect foundation cool vanilla by no...	no7 stay perfect foundation cool vanilla by no7	no7	all beauty
8279996397	imagen bendita por su santidad our lady of cha...	imagen bendita por su santidad our lady of cha...	chango	all beauty

asin	text	title	brand	main_cat
9197882747	hall of femmes: lella vignelli (hall of femmes...	hall of femmes: lella vignelli (hall of femmes)	oyster press	all beauty
...	...	...	...	...
B01HHFHGES	xiaokong women's high low hem sleeveless flora...	xiaokong women's high low hem sleeveless flora...	xiaokong	books
B01HHX5AF2	duxa women's working my adipose off symbol gra...	duxa women's working my adipose off symbol gra...	duxa	books
B01HHZYKYW	aliixun2 unisex the 2016 rio de janeiro olympi...	aliixun2 unisex the 2016 rio de janeiro olympi...	aliixun2	books
B01HIUH2AK	busoni : konzertstuck fur klavier mit orcheste...	busoni : konzertstuck fur klavier mit orchester	ferruccio busoni	books
B01HIXTMKA	thin red line flag bling baseball cap distress...	thin red line flag bling baseball cap distress...	elivata	books

3679356 rows × 4 columns

In [4]:
```
cleaning.info(df)
```

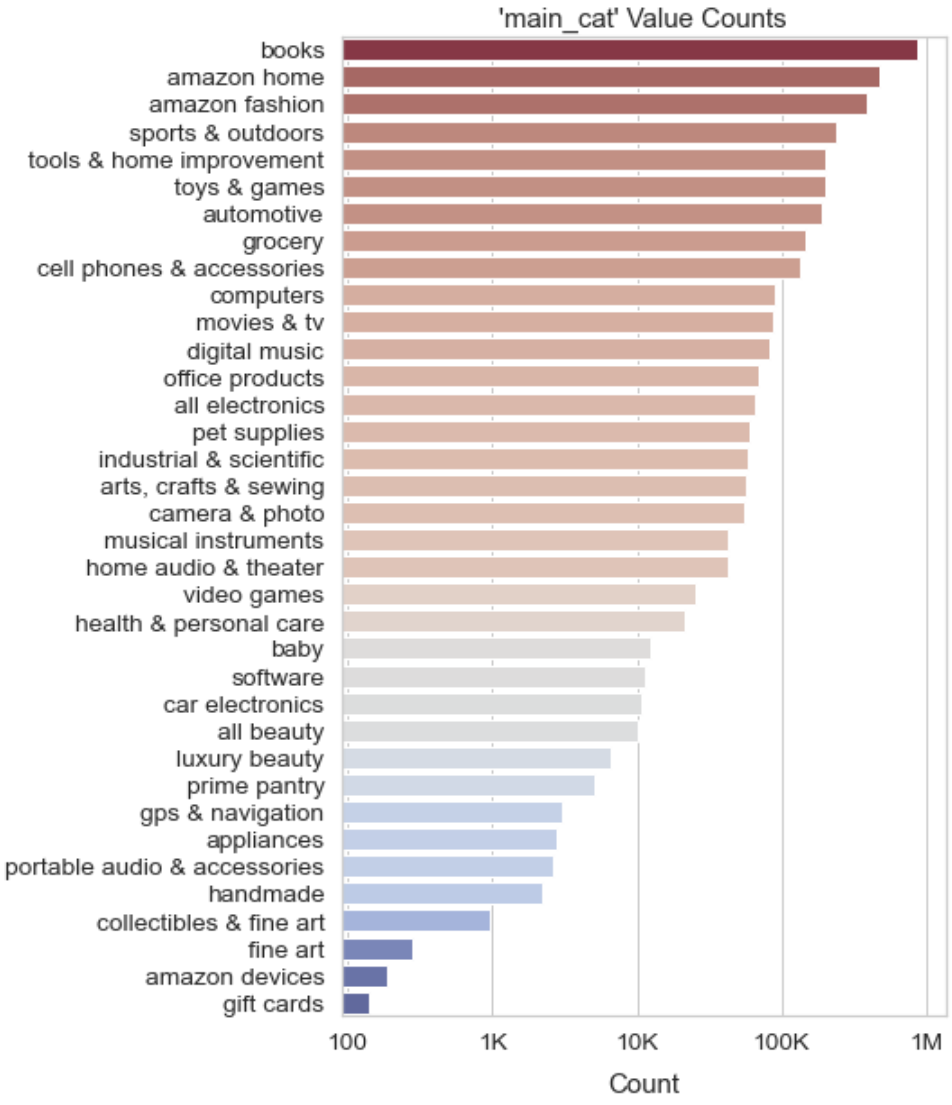
Out[4]:

	null	null_%	uniq	uniq_%	dup	dup_%
text	0	0.0	3679356	100.00	0	0.0
title	0	0.0	3659460	99.46	0	0.0
brand	0	0.0	817732	22.22	0	0.0
main_cat	0	0.0	36	0.00	0	0.0

No duplicates or nulls (at least ostensibly). Looks good.

Next I plot the main categories to get a sense of the balance.

In [5]:
```
ax = plotting.plots.countplot(df.loc[:, "main_cat"], size=(6, 10), log_scale=True)
ax.xaxis.set_major_formatter(plotting.big_number_formatter())
```



The classes are very imbalanced, which is why I had to plot them on a logarithmic scale. Books are at the top.

Refining the Brands

I begin with the brand terms, which will be very important for the classifier.

In [6]:
```
empty_brand = df.loc[df.brand.str.fullmatch("\s*")]
empty_brand
```

Out[6]:

asin	text	title	brand	main_cat
B000050B67	norelco 5655x deluxe advantage wet/dry cordles...	norelco 5655x deluxe advantage wet/dry cordles...		all beauty
B000050B69	norelco t7500 deluxe cord/cordless rechargeabl...	norelco t7500 deluxe cord/cordless rechargeabl...		all beauty
B000050FDP	braun 6520 flex integral ultra speed rechargea...	braun 6520 flex integral ultra speed rechargea...		all beauty
B000052YD8	scope original mint mouthwash 50.7 fl oz. scop...	scope original mint mouthwash 50.7 fl oz		all beauty

asin	text	title	brand	main_cat
B00005336W	dove nutrium nutrient beads nourishing body wa...	dove nutrium nutrient beads nourishing body wa...		all beauty
...	...	...	...	...
B01H0UDFXQ	marc lawrence's 2016 playbook football preview...	marc lawrence's 2016 playbook football preview		books
B01H3U5GNA	american original soft and cozy coastal stripe...	american original soft and cozy coastal stripe...		books
B01H87P6YM	hitachi 43in 2160p 120hz 4k uhd led tv. the ne...	hitachi 43in 2160p 120hz 4k uhd led tv		books
B01HB59462	hanging chaise lounger chair arc stand air por...	hanging chaise lounger chair arc stand air por...		books
B01HCAUSG6	elizabeth jane howard cazalet chronicles 5 boo...	elizabeth jane howard cazalet chronicles 5 boo...		books

146476 rows × 4 columns

Looks like there are over about 1.6 million empty brand entries. I'll replace those with 'none' for now.

In [7]:
```
df.loc[empty_brand.index, "brand"] = "none"
df.loc[df.brand.str.fullmatch("\s*")]
```

Out[7]:

asin	text	title	brand	main_cat

In [8]:
```
del empty_brand
```

Next, I'm going to tokenize the brands using the default Scikit-Learn tokenizer regex (r'\b\w\w+\b'). This is the default value for 'token_pattern' in `CountVectorizer`, `HashingVectorizer`, and `TfidfVectorizer`.

I like the default Scikit-Learn tokenizer because it ignores punctuation and grabs sequences of 2 or more word characters (within word boundaries). It strips out most of the noise in a single stroke.

I define a function below using this pattern.

In [9]:
```
def sklearn_tokenize(docs, n_jobs=None):
    pattern = re.compile(r"\b\w\w+\b")
    return lang.process_strings(docs, pattern.findall, n_jobs=n_jobs)
```

It's a short function, but a powerful one. It's **polymorphic**, meaning that it it behaves differently depending on the input type. If `docs` is a string, it simply returns the list of tokens. However, if `docs` is a container type like `Series` or `DataFrame`, it applies the tokenizer to every element. It also features **multiprocessing**, which can dramatically increase performance on large datasets.

Like many functions you'll see this notebook, it uses the infrastructure provided by my low-level function `lang.process_strings`.

In [10]:
```
help(lang.process_strings)
```

```
Help on function process_strings in module tools.language.utils:

process_strings(strings: Union[str, Iterable[str]], func: Callable[[str], Any], n_jobs: int = None,
**kwargs) -> Any
    Apply `func` to a string or iterable of strings (elementwise).

    Most string filtering/processing functions in the language module
    are polymorphic, capable of handling either a single string or an
    iterable of strings. Whenever possible, they rely on this generic
    function to apply a callable to string(s). This allows them to
    behave polymorphically and take advantage of multiprocessing while
    having a simple implementation.

    This is a single dispatch generic function, meaning that it consists
    of multiple specialized sub-functions which each handle a different
    argument type. When called, the dispatcher checks the type of the first
    positional argument and then dispatches the sub-function registered
    for that type. In other words, when the function is called, the call
    is routed to the appropriate sub-function based on the type of the first
    positional argument. If no sub-function is registered for a given type,
    the correct dispatch is determined by the type's method resolution order.
    The function definition decorated with `@singledispatch` is registered for
    the `object` type, meaning that it's the dispatcher's last resort.

    The most important sub-function is the list dispatch, where
    multiprocessing is implemented via Joblib. Every other sub-function
    (besides str) routes data there for multiprocessing.

    Parameters
    ----------
    strings : str, iterable of str
        String(s) to map `func` over. Null values are ignored.
    func : Callable
        Callable for processing `strings`.
    n_jobs: int
        The maximum number of concurrently running jobs. If -1 all CPUs are used.
        If 1 or None is given, no parallel computing code is used at all. For n_jobs
        below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but
        one are used. Defaults to None.
    **kwargs
        Keyword arguments for `func`.

    Returns
    -------
    Any
        Processed string(s), same container type as input.
```

Another lower-level function you'll see is `lang.chain_processors`, which allows me to apply a function-pipeline to string(s). This is especially useful when I need to tokenize some strings, do something, and then detokenize them. In the following cell, I coerce the brand names to ASCII, tokenize them, and then immediately detokenize them. I want to reduce the number of unique values by reducing the number of near-duplicates.

In [11]:
```
# Function pipeline given as a list
steps = [lang.force_ascii, sklearn_tokenize, "_".join]

# Apply pipeline using all available cores
df["brand"] = lang.chain_processors(df.loc[:, "brand"], steps, n_jobs=-1)

df["brand"].value_counts()
```

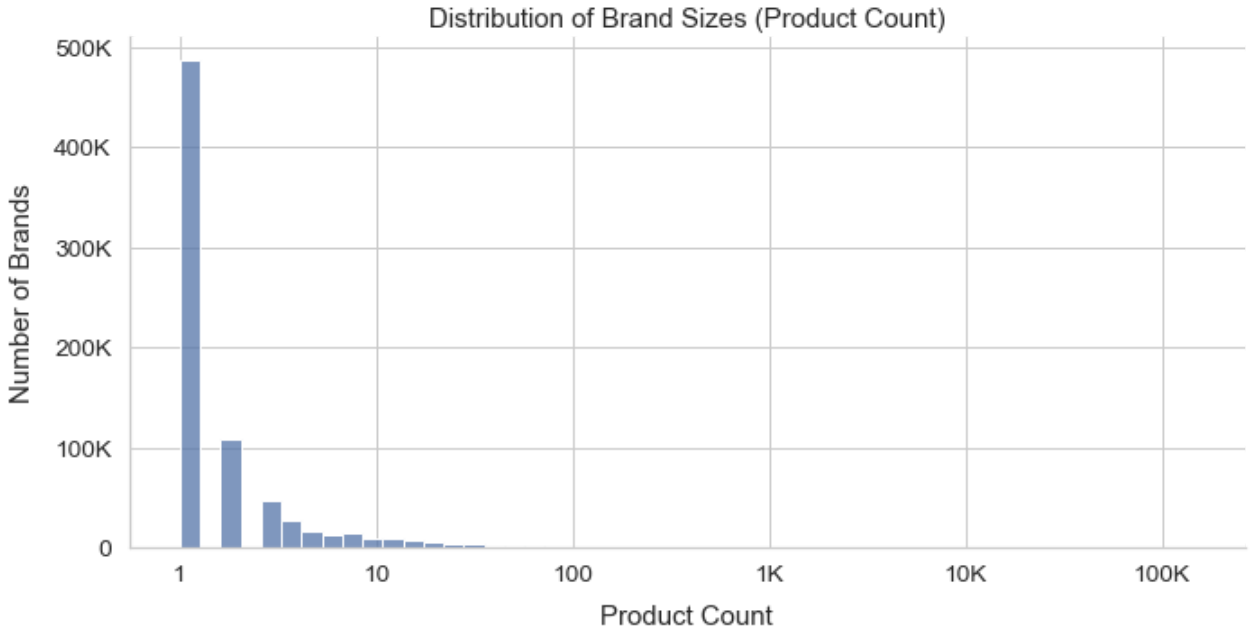
Out[11]:
```
none                  146481
generic                10173
yu_gi_oh                7075
magic_the_gathering     6488
nike                    5757
```

```
                          ...
bernard_clark_jr            1
ben_griffith                1
popupo                      1
fendi_kids                  1
to_speak_of_wolves          1
Name: brand, Length: 768000, dtype: int64
```

Over 790,000 unique brands! I'm going to use the brands as ngram phrases in the text, but I suspect most of these are too obscure to be useful. One-off brand terms are almost completely useless for a term-frequency-based classification, since they'll only appear in one vector.

In [12]:
```
def plot_brand_size_dist(data, brand_col="brand", bins=50, log_scale=True):
    brand_counts = data[brand_col].value_counts()
    g = sns.displot(
        data=brand_counts.to_frame(),
        x=brand_col,
        log_scale=log_scale,
        bins=bins,
        aspect=2,
    )
    g.axes[0, 0].set(
        xlabel="Product Count",
        ylabel="Number of Brands",
        title="Distribution of Brand Sizes (Product Count)",
    )
    g.axes[0, 0].yaxis.set_major_formatter(plotting.big_number_formatter())
    g.axes[0, 0].xaxis.set_major_formatter(plotting.big_number_formatter())
```

In [13]:
```
plot_brand_size_dist(df)
```



Over 500,000 brands have just one instance. I'm not interested in those.

In order to engineer a solid dataset for product classification, I'm going to keep only the biggest brands in each main category. That ought to get rid of most of the brand noise and make room for some signal.

For each main category, (with some exceptions) I keep only the top 1% of brands. Again, I'm measuring brand size in terms of number of products. Main categories with less than 1,000 samples are exempt from the pruning.

Furthermore, all *none* entries will be dropped except in categories which are almost entirely *none* (in order to preserve the categories). The 'fine art' category, for instance, is ~99% *none*, presumably due to the unique nature of the products. Currently Amazon does show brands for fine art products, but that may have been different in 2018, when this data was collected.

In [14]:

```
# Drop brands below this quantile
qcut = 0.99

# Keep nulls if they dominate the category
null_dom_thresh = 0.95

# Don't prune small categories
protected_thresh = 1000

pruned_df = []

for cat, group in df.groupby("main_cat"):

    # Ignore small categories
    if len(group) <= protected_thresh:
        pruned_df.append(group)
        continue

    # Drop nulls if freq under `null_dom_thresh`
    brand_counts = group["brand"].value_counts(1)
    if brand_counts.get("none", 1.0) < null_dom_thresh:
        group = group.loc[group.brand != "none"]

    print(cat)
    print("-" * len(cat))

    # Drop brands below quantile `qcut`
    group = utils.prune_categories(group, "brand", qcut=qcut)

    print("\n")
    pruned_df.append(group)

pruned_df = pd.concat(pruned_df)
```

```
all beauty
----------
                    Status  Support
crystalage        retained       89
makingcosmetics   retained       80
kroo              retained       67
crest             retained       64
vktech            retained       60
...                    ...      ...
aero_cosmetics     dropped        1
lobob              dropped        1
pinkiou            dropped        1
artisano_designs   dropped        1
stockhausen        dropped        1

[5175 rows x 2 columns]


all electronics
---------------
                      Status  Support
sony                retained     1202
panasonic           retained     1137
samsung             retained      910
hp                  retained      635
startech            retained      522
...                      ...      ...
amazing_accessory    dropped        1
```

```
leadtry             dropped        1
vodavi              dropped        1
sevenoak            dropped        1
philips_uhp         dropped        1

[12228 rows x 2 columns]


amazon fashion
--------------
                         Status  Support
amazon_collection     retained     4718
invicta               retained     3654
nike                  retained     3092
adidas                retained     2731
seiko                 retained     2203
...                        ...      ...
nowhy                  dropped        1
uspa                   dropped        1
lyt_lanx               dropped        1
rocky_and_bullwinkle   dropped        1
closetmaid             dropped        1

[30847 rows x 2 columns]


amazon home
-----------
                             Status  Support
3drose                     retained     2618
trademark_fine_art         retained     1941
tree_free_greetings        retained     1582
generic                    retained     1536
home_decorators_collection retained     1436
...                             ...      ...
hamiledyi                   dropped        1
rock_santa_collectibles     dropped        1
general_tools_instruments   dropped        1
masters_of_the_universe     dropped        1
wildlife_sciences           dropped        1

[67615 rows x 2 columns]


appliances
----------
                       Status  Support
frigidaire           retained      169
akdy                 retained       85
whirlpool            retained       84
broan                retained       83
avanti               retained       81
...                       ...      ...
envirocon             dropped        1
hqf                   dropped        1
woodcraft_woodshop    dropped        1
englander             dropped        1
talking_products      dropped        1

[548 rows x 2 columns]


arts, crafts & sewing
---------------------
                     Status  Support
lantern_press     retained     1030
3drose            retained      829
beadaholique      retained      795
sizzix            retained      696
spellbinders      retained      597
...                    ...      ...
unsetgems          dropped        1
```

```
ghong             dropped        1
for_mom           dropped        1
kirinstores       dropped        1
premium_knitter   dropped        1

[9517 rows x 2 columns]


automotive
----------
                                 Status  Support
evan_fischer                   retained     4022
curt_manufacturing             retained     2353
autoandart                     retained     2190
tuningpros                     retained     1928
rareelectrical                 retained     1505
...                                 ...      ...
jump_start_it                   dropped        1
eaglecollector83                dropped        1
primesource_building_products   dropped        1
silhouette_america              dropped        1
kal_equip                       dropped        1

[15371 rows x 2 columns]


baby
----
                     Status  Support
fisher_price       retained      184
gund               retained      162
kids_preferred     retained      152
stephan_baby       retained      126
carter             retained      123
...                     ...      ...
trendy_kitty        dropped        1
njseller_cn         dropped        1
belsen              dropped        1
raphael_rozen       dropped        1
larivia_essential   dropped        1

[3806 rows x 2 columns]


books
-----
                   Status  Support
various          retained      619
zondervan        retained      385
fodor            retained      258
dk_publishing    retained      236
hal_leonard      retained      225
...                   ...      ...
harjit_singh      dropped        1
mary_garbe        dropped        1
barbara_bowman    dropped        1
alan_finn         dropped        1
hendelie          dropped        1

[469541 rows x 2 columns]


camera & photo
--------------
               Status  Support
canon        retained     1467
sony         retained     1210
nikon        retained     1158
neewer       retained     1150
manfrotto    retained      600
...               ...      ...
alienono      dropped        1
```

```
lary_intel   dropped        1
gigtube      dropped        1
swivl        dropped        1
add_motor    dropped        1

[6608 rows x 2 columns]


car electronics
---------------
                      Status  Support
pyle                retained      464
boss_audio_systems  retained      348
kicker              retained      266
metra               retained      210
pioneer             retained      186
...                      ...      ...
winnereco            dropped        1
app_tronics          dropped        1
blackbox_guard       dropped        1
gleim                dropped        1
minder_research      dropped        1

[2218 rows x 2 columns]


cell phones & accessories
-------------------------
                  Status  Support
empire          retained     3299
amzer           retained     2193
mybat           retained     1845
generic         retained     1784
samsung         retained     1609
...                  ...      ...
beartybohochic   dropped        1
caddie_buddy     dropped        1
coolmate         dropped        1
featheria        dropped        1
bubba            dropped        1

[18293 rows x 2 columns]


computers
---------
                             Status  Support
hp                         retained     2408
dell                       retained     2138
acer                       retained     1440
generic                    retained     1310
asus                       retained     1166
...                             ...      ...
epraizer                    dropped        1
amzan                       dropped        1
enegitech                   dropped        1
racing_electronics          dropped        1
personalized_laptop_sleeve  dropped        1

[12867 rows x 2 columns]


digital music
-------------
                           Status  Support
various_artists          retained     4516
various                  retained     1104
ludwig_van_beethoven     retained      193
johann_sebastian_bach    retained      174
wolfgang_amadeus_mozart  retained      153
...                           ...      ...
larkin_grimm              dropped        1
```

```
marshmellow_coast         dropped        1
nightbeast                dropped        1
tommy_february            dropped        1
marian_anderson           dropped        1

[43252 rows x 2 columns]


gps & navigation
----------------
                          Status  Support
gomadic                 retained      264
garmin                  retained      128
pcprofessional          retained       76
tenq                    retained       66
duragadget              retained       49
...                          ...      ...
delphi                   dropped        1
jlm_security_products    dropped        1
nextar                   dropped        1
interphone               dropped        1
vertex_standard          dropped        1

[849 rows x 2 columns]


grocery
-------
                            Status  Support
black_tie_mercantile       retained     1310
mccormick                  retained      587
jelly_belly                retained      541
bob_red_mill               retained      528
starbucks                  retained      489
...                             ...      ...
plainville_farms            dropped        1
grand_cru_international     dropped        1
labadie_bakery              dropped        1
classic_provisions_spices   dropped        1
temeraire                   dropped        1

[24660 rows x 2 columns]


handmade
--------
                          Status  Support
none                    retained     2212
unique                   dropped        6
marvel                   dropped        3
monster                  dropped        2
apple                    dropped        2
brenda_elaine_jewelry    dropped        1
lego                     dropped        1


health & personal care
----------------------
                               Status  Support
maxiaids                     retained      139
liliane_memorials            retained      126
crystalage                   retained      115
terravita                    retained       89
big_dot_of_happiness         retained       88
...                               ...      ...
airfit_p10_headagear_for_her   dropped        1
philips_respironics            dropped        1
genuine_amber                  dropped        1
body_prop                      dropped        1
stockhausen                    dropped        1

[10323 rows x 2 columns]
```

home audio & theater

	Status	Support
sony	retained	1137
redi_remote	retained	884
monster	retained	632
philips	retained	572
monoprice	retained	432
...	...	...
iasus	dropped	1
crazy_controllerz	dropped	1
itd_itanda	dropped	1
na_santa	dropped	1
exuun	dropped	1

[7996 rows x 2 columns]

industrial & scientific

	Status	Support
small_parts	retained	1753
vxb	retained	728
compliancesigns	retained	672
uxcell	retained	660
vestil	retained	466
...	...	...
rollpak	dropped	1
signode	dropped	1
spj	dropped	1
insma_be_interesting_be_smart	dropped	1
eiko	dropped	1

[10386 rows x 2 columns]

luxury beauty

	Status	Support
none	retained	6623
archipelago_botanicals	dropped	2
oribe	dropped	1
h2o_beauty	dropped	1
phyto	dropped	1
red_flower	dropped	1
bliss	dropped	1
zwilling_henckels	dropped	1

movies & tv

	Status	Support
various	retained	2011
sinister_cinema	retained	177
levar_burton	retained	106
na	retained	97
the_ambient_collection	retained	87
...	...	...
larkin	dropped	1
amy_eastwood_and_tara_lee	dropped	1
justina_machado	dropped	1
chris_finch	dropped	1
edwardo_ferreira	dropped	1

[33652 rows x 2 columns]

musical instruments

	Status	Support

```
seismic_audio              retained     1012
fender                     retained      934
yamaha                     retained      695
kmise                      retained      420
ibanez                     retained      350
...                             ...      ...
manu                       dropped         1
fatboy_sounds              dropped         1
bg                         dropped         1
paco_natural_wood_maracas  dropped         1
cyclone                    dropped         1

[6266 rows x 2 columns]
```

```
office products
---------------
                       Status  Support
at_glance            retained     1407
avery                retained     1070
displays2go          retained      808
ld_products          retained      803
moleskine            retained      702
...                       ...      ...
anitoon               dropped        1
deltron_essentials    dropped        1
smith_victor          dropped        1
thinp                 dropped        1
northern_response     dropped        1

[12880 rows x 2 columns]
```

```
pet supplies
------------
                         Status  Support
pets                   retained      411
casual_canine          retained      373
zack_zoey              retained      357
east_side_collection   retained      337
alfie                  retained      326
...                         ...      ...
panasiastore            dropped        1
whistles                dropped        1
smart_ato_micro         dropped        1
quick_clean             dropped        1
ideal_fast_fit          dropped        1

[12236 rows x 2 columns]
```

```
portable audio & accessories
----------------------------
                        Status  Support
gomadic               retained      132
skinit                retained       59
generic               retained       53
griffin_technology    retained       42
handhelditems         retained       37
...                        ...      ...
samrick                dropped        1
insanix                dropped        1
mangotek               dropped        1
enersound              dropped        1
techtoo                dropped        1

[1211 rows x 2 columns]
```

```
prime pantry
------------
                 Status  Support
```

```
oreal_paris    retained      133
quaker         retained       86
nature_made    retained       83
neutrogena     retained       79
garnier        retained       76
...                 ...      ...
tiger_balm      dropped        1
apple_jacks     dropped        1
sugar_company   dropped        1
mueller         dropped        1
ardell          dropped        1

[1187 rows x 2 columns]
```

```
software
--------
                       Status  Support
microsoft             retained      416
encore                retained      412
topics_entertainment  retained      328
infiniteskills        retained      218
intuit                retained      207
...                        ...      ...
dr_david_shormann      dropped        1
babylon                dropped        1
evidentia_software     dropped        1
snapware               dropped        1
administaff_hrtools    dropped        1

[2501 rows x 2 columns]
```

```
sports & outdoors
-----------------
                Status  Support
nike          retained     2584
adidas        retained     2206
under_armour  retained     2020
majestic      retained     1846
speedo        retained     1297
...                ...      ...
goose_co       dropped        1
drop_shades    dropped        1
mtt_pl         dropped        1
throw_raft     dropped        1
jimblaster     dropped        1

[36243 rows x 2 columns]
```

```
tools & home improvement
------------------------
                             Status  Support
kohler                      retained     2152
leviton                     retained     1778
dewalt                      retained     1581
moen                        retained     1454
kichler                     retained     1392
...                              ...      ...
44_llc                       dropped        1
st_patrick_day_maibox_cover  dropped        1
pro_led_shower               dropped        1
bellota                      dropped        1
technonyx                    dropped        1

[25015 rows x 2 columns]
```

```
toys & games
------------
                             Status  Support
```

```
yu_gi_oh                       retained     7068
magic_the_gathering            retained     6487
pokemon                        retained     3124
mattel                         retained     2541
hasbro                         retained     2441
...                                 ...      ...
ouran_high_host_club           dropped        1
phd_productions                dropped        1
princess_wands_party_favors    dropped        1
time_out_pad                   dropped        1
bubba                          dropped        1

[22034 rows x 2 columns]


video games
-----------
                              Status  Support
electronic_arts             retained     1324
activision                  retained      930
nintendo                    retained      864
ubisoft                     retained      851
sony                        retained      622
...                              ...      ...
for_ps3                     dropped        1
crystal_vision_technology   dropped        1
rivalware_entertainment     dropped        1
beautifun_games_sl          dropped        1
napland_games               dropped        1

[3187 rows x 2 columns]
```

In [15]:

```
size_reduct = 1 - (len(pruned_df) / len(df))
print(f"Reduced n_samples by {size_reduct:.0%}")
pruned_df
```

```
Reduced n_samples by 71%
```

Out[15]:

asin	text	title	brand	main_cat
B000050AUH	philips sonicare standard brush head. sonic wa...	philips sonicare standard brush head	philips_sonicare	all beauty
B000050B62	norelco 5841xl deluxe reflex action cord/cordl...	norelco 5841xl deluxe reflex action cord/cordl...	norelco	all beauty
B000050B63	norelco 6826xl quadra action cord/cordless rec...	norelco 6826xl quadra action cord/cordless rec...	norelco	all beauty
B000050B64	norelco 6865xl quadra action cord/cordless rec...	norelco 6865xl quadra action cord/cordless rec...	norelco	all beauty
B000050B65	norelco 6885xl deluxe quadra action cord/cordl...	norelco 6885xl deluxe quadra action cord/cordl...	norelco	all beauty
...	...	...	...	...
B01HH6JEOC	the sims 4 kids room stuff [online game code]....	the sims 4 kids room stuff [online game code]	electronic_arts	video games
B01HIU43S4	1k games sega mega drive game console with wir...	1k games sega mega drive game console with wir...	sega	video games
B01HIZF83S	bioshock: the collection - playstation 4. retu...	bioshock: the collection - playstation 4	2k	video games

asin	text	title	brand	main_cat
B01HJ149LI	god eater resurrection - ps vita [digital code...	god eater resurrection - ps vita [digital code]	bandai	video games
B01HJ14FDA	jojo eyes of heaven complete bundle - ps4 [dig...	jojo eyes of heaven complete bundle - ps4 [dig...	bandai	video games

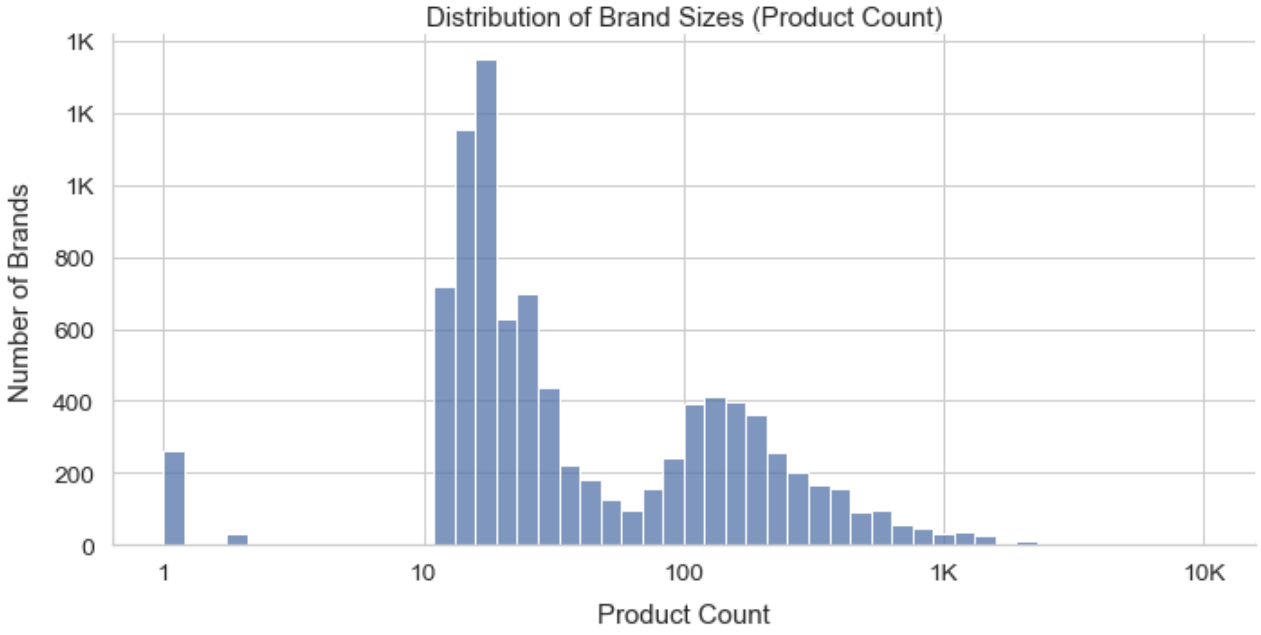
1068382 rows × 4 columns

In [16]:
```
pruned_df.brand.nunique()
```

Out[16]: 9109

This is an enormous dataset, so the top 1% of brands in each category leaves me with ~9k brands. That's a fairly large, yet manageable, number.

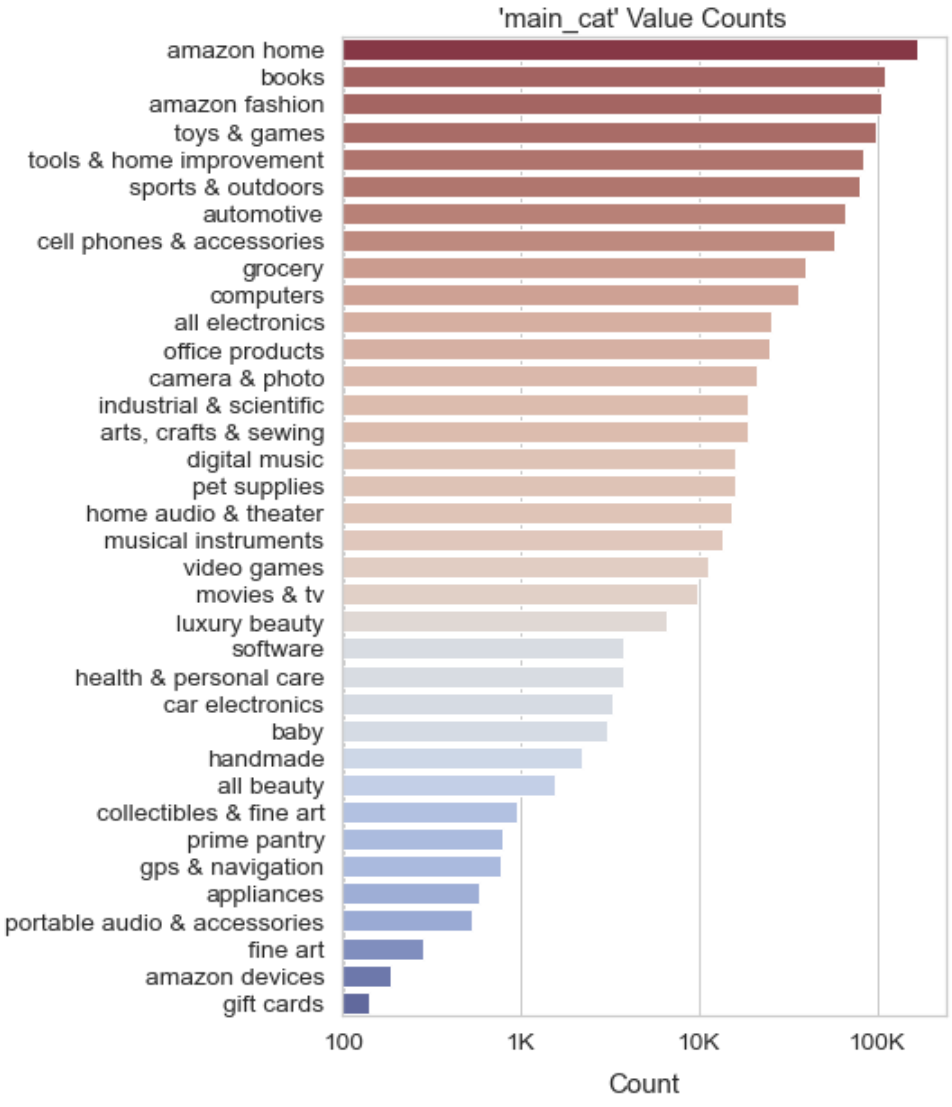
In [17]:
```
plot_brand_size_dist(pruned_df)
```



Although I only took the top 1% of brands in each category, most of the brands have 10-50 products. That seems like a healthy number.

The histogram also shows that there are ~400 single-product brands from the protected categories. That's fine. Those terms won't be very useful, but there's plenty of other useful information in those categories.

In [18]:
```
ax = plotting.plots.countplot(
    pruned_df.loc[:, "main_cat"], log_scale=True, size=(6, 10)
)
ax.xaxis.set_major_formatter(plotting.big_number_formatter())
```



In [19]:
```
df = pruned_df
df.head()
```

Out[19]:

asin	text	title	brand	main_cat
B000050AUH	philips sonicare standard brush head. sonic wa...	philips sonicare standard brush head	philips_sonicare	all beauty
B000050B62	norelco 5841xl deluxe reflex action cord/cordl...	norelco 5841xl deluxe reflex action cord/cordl...	norelco	all beauty
B000050B63	norelco 6826xl quadra action cord/cordless rec...	norelco 6826xl quadra action cord/cordless rec...	norelco	all beauty
B000050B64	norelco 6865xl quadra action cord/cordless rec...	norelco 6865xl quadra action cord/cordless rec...	norelco	all beauty
B000050B65	norelco 6885xl deluxe quadra action cord/cordl...	norelco 6885xl deluxe quadra action cord/cordl...	norelco	all beauty

I check the brands of 'arts, crafts, & sewing' as a basic sanity check.

In [20]:
```
df.groupby("main_cat").get_group("arts, crafts & sewing").brand.value_counts()
```

Out[20]:
```
lantern_press               1030
3drose                       829
beadaholique                 795
sizzix                       696
spellbinders                 597
                            ...
moda                          86
unique_wood_shapes            85
crafter_companion             85
vogue_fabrics                 83
sparkles_make_it_special      82
Name: brand, Length: 96, dtype: int64
```

Seems to be a healthy number.

Engineering the Vocabulary

I engineer the model's vocabulary by preprocessing the text, developing a custom tokenizer, and constructing multi-word expressions. The purpose of the model is to assimilate novel products into Amazon's existing classification scheme. If the vocabulary is to serve that purpose, it will require some *a priori* decision-making.

To be clear, I don't intend to feed `TfidfVectorizer` a fixed vocabulary, because that would restrict my ability to tune the vocabulary later on. I plan on letting the vectorizer learn the vocabulary on its own, but I want it to learn the *right* vocabulary.

Before making any decisions, I define a function for creating a vocabulary from a sample of the corpus.

In [21]:
```python
def build_sample_vocab(
    corpus,
    n_docs,
    *,
    tokenizer=None,
    sortby="length",
    random_state=363,
    n_jobs=-1,
):
    # Get sample of corpus
    if n_docs is None:
        samp_corpus = corpus
    else:
        samp_corpus = corpus.sample(n_docs, random_state=random_state)

    # Tokenize
    if tokenizer is None:
        # Check if already tokenized
        if not pd.api.types.is_list_like(corpus.iloc[0]):
            samp_corpus = lang.space_tokenize(samp_corpus, n_jobs=n_jobs)
    else:
        samp_corpus = lang.process_strings(samp_corpus, tokenizer, n_jobs=n_jobs)

    # Build vocabulary
    vocab = samp_corpus.explode().value_counts()
    vocab = vocab.reset_index(name="freq").rename(columns={"index": "term"})

    # Add additional stats
    vocab["length"] = vocab.loc[:, "term"].str.len()
    vocab["uniq_ratio"] = vocab.loc[:, "term"].map(lang.uniq_ratio)

    return vocab.sort_values(sortby, ascending=False)
```

Optimize the Tokenizer?

There are many tokenizers on the market. Shouldn't I grid-search over them all to maximize my accuracy score?

No. The choice of tokenizer requires some *human* intelligence. The purpose of the model is to classify new products into Amazon's existing categories. High test scores do not always indicate a better model—sometimes, they reflect overfitting.

Take for example `nltk.wordpunct_tokenize`. This tokenizer captures arbitrarily long sequences of punctuation in addition to normal words.

In [22]:
```
# Get small sample of corpus
wordpunct_vocab = build_sample_vocab(
    df.loc[:, "text"], 10 ** 4, tokenizer=nltk.wordpunct_tokenize
)

# Slice out some punctuation
wordpunct_vocab.loc[wordpunct_vocab.term.str.startswith("!"), "term"].head(10)
```

Out[22]:
```
24240    !~~~~~~~~~~~~~~~~~
47011           !!!!!!!!!!.
50243             !!!~~~~~~
59635               !!!!!!.
61963                 !!!!!
65858                 !!!!.
42982                  !").
33264                  !**.
41184                  !"".
23630                  !!!!
Name: term, dtype: object
```

In [23]:
```
del wordpunct_vocab
```

The problem is that these sequences are *artifacts* of this particular dataset. Suppose that the model learns to associate '##############' with the 'automotive' category and '&&&&&&&&&!?!?!?' with 'baby'. That might be a real pattern in this particular dataset, but it's useless in general.

I begin with the the Scikit-Learn default tokenizer that I used on the brands. It captures sequences of two or more alphanumeric characters within a word boundary. It ignores punctuation and single letters, which results in very clean tokens. For example, 'Frank's' becomes 'Frank'. It's also lightning fast because it's just a regular expression.

In [24]:
```
# Get small sample of corpus
sklearn_vocab = build_sample_vocab(
    df.loc[:, "text"], 10 ** 4, tokenizer=sklearn_tokenize
)
sklearn_vocab.sort_values("length", ascending=False, inplace=True)

# Slice out numeric strings, sort by length
sklearn_vocab.loc[sklearn_vocab.term.str.isnumeric(), "term"].head(10)
```

Out[24]:
```
49818    0020473440003353
59004      10763649016548
65876      10763649016562
45603      10763649016517
45625      10763649016531
48547       9780785820345
45802       9781569755785
57500       0000772088527
44383       4051771576894
42324       9780785818564
Name: term, dtype: object
```

In [25]:
```
del sklearn_vocab
```

One problem with the Scikit-Learn tokenizer is that it produces code-like numeric sequences. My goal is not to create a model which picks up on ultra-rare ID numbers and codes for specific products, but to create a model which recognizes broad patterns in **natural language**.

I define my own tokenizer in the cell below.

In [26]:
```
def my_tokenize_1(docs, n_jobs=None):
    pattern = re.compile(r"(?i)\b[a-z_]{2,}\b")
    return lang.process_strings(docs, pattern.findall, n_jobs=n_jobs)
```

The tokenizer I created above captures sequences of two or more alphabetic ASCII characters (plus underscore) `'[a-z_]{2,}'` within word boundaries `'\b'` while ignoring case `(?i)`. It ignores non-ASCII characters including accented letters, which I deliberately conflate with their non-accented counterparts.

In [27]:
```
df["text"] = lang.force_ascii(df.loc[:, "text"], n_jobs=-1)
df["text"].head()
```

Out[27]:
```
asin
B000050AUH    philips sonicare standard brush head. sonic wa...
B000050B62    norelco 5841xl deluxe reflex action cord/cordl...
B000050B63    norelco 6826xl quadra action cord/cordless rec...
B000050B64    norelco 6865xl quadra action cord/cordless rec...
B000050B65    norelco 6885xl deluxe quadra action cord/cordl...
Name: text, dtype: object
```

One thing my tokenizer doesn't address is tokens that are too long.

In [28]:
```
samp_vocab = build_sample_vocab(df.loc[:, "text"], 10 ** 4, tokenizer=my_tokenize_1)

samp_vocab.head(10)
```

Out[28]:

	term	freq	length	uniq_ratio
50267	lovepoemsandmorefromtheheartandsoulofman	1	40	0.375000
37094	enclosurefeaturesalightweightyet	1	32	0.500000
51153	atramentizedpolishedinsulated	1	29	0.517241
29145	the_strictly_mint_card_co_inc	2	29	0.517241
28554	green_compliance_certificate	2	28	0.500000
45458	mmpliersheadhandlesstripping	1	28	0.464286
34329	atramentizedpolishedplastic	1	27	0.555556
45239	ninjagocoleminifigurebuilt	1	26	0.576923
44120	instructionsmarkingoptions	1	26	0.500000
34066	pieceimxcompositefeatures	1	25	0.520000

As you can see, most of these 20+ character strings are just idiosyncratic noise. Take for example the term 'sherryguzzlinggranny':

In [29]:
```
granny_book = df.loc[df.text.str.contains("sherryguzzlinggranny")].iloc[0]
```

```
display(granny_book)
granny_book.text
```

```
text        queen of babble gets hitched. when last seen, ...
title                          queen of babble gets hitched
brand                                              meg_cabot
main_cat                                               books
Name: 006085202X, dtype: object
```

Out[29]:
```
"queen of babble gets hitched. when last seen, the irrepressible lizzie nichols was canoodling with
chaz after she and luke, chaz's best friend, broke up ( queen of babble in the big city , 2007). no
wshocker alertluke returns to new york and slips a three-carat diamond engagement ring on her finge
r. lizzie accepts even though she's still all googly over chaz, who bluntly warns lizzie that luk
e's all about luke and couldn't love her the way he does. lizzie, a wedding dress restorer and budd
ing designer specializing in wedding garb, faces a hives-inducing decision: dump rich luke, who wan
ts to be an investment banker in paris, and hook up with chaz, who wants to teach? or should she ma
rry luke and ditch new york for paris? and then there's the matter of her burgeoning design busines
s, helped along by ava geck, a paris hiltonlike celebrity heiress. cabot takes full advantage of th
e material, delivering her trademark wit, sharp banter and lively antics from the first page. fans
of the series have another one to savor. (july) copyright reed business information, a division of
reed elsevier inc. all rights reserved. lizzie nichols should be the happiest girl in the universe.
the phone has been ringing off the hook eversince a wedding gown she restored appeared inthe new yo
rk times. her wealthy ex-boyfriend, luke de villier, has decided their breakup was a mistake and ha
s finally proposed.but lizziemade the mistake of sleeping with chaz, lukes best friend, during the
brief hiatus in her relationship with luke, and now she spends more time thinking about chaz thanhe
r fianc. in addition to an angst-ridden heroine and a loyal, waiting-in-the-wings hero, cabots nove
lis filled with unforgettable, quirky characters. theres lizziesbad-mouthed, cooking sherryguzzling
granny; a spoiled, self-centered fianc; a best friend who is a newly outed lesbian; and a high-stru
ngassistant. celebrities demanding a lizzie nichols creation include afamiliar, gum-popping, red-ca
rpet regular always accompanied byher shivering chihuahua and bodyguard. to add to the fun, each ch
apter begins with wedding trivia and advice.cabot, author of the humorous heather wells mysteriesas
well as the wildly popular princess diaries series, has once again shown that she is a master at en
tertaining and amusingreaders. --shelley mosley. meg cabot."
```

The term 'sherryguzzlinggranny' comes from the book *Queen of Babble Gets Hitched*. The description (above) contains a number of similar typos, such as 'lizziesbad-mouthed', 'high-strungassistant', and 'amusingreaders'. I have no qualms about filtering these out.

Next I get a larger sample and examine the length distribution.

In [30]:
```
samp_vocab = build_sample_vocab(df.loc[:, "text"], 10 ** 5, tokenizer=my_tokenize_1)

samp_vocab.sample(10)
```

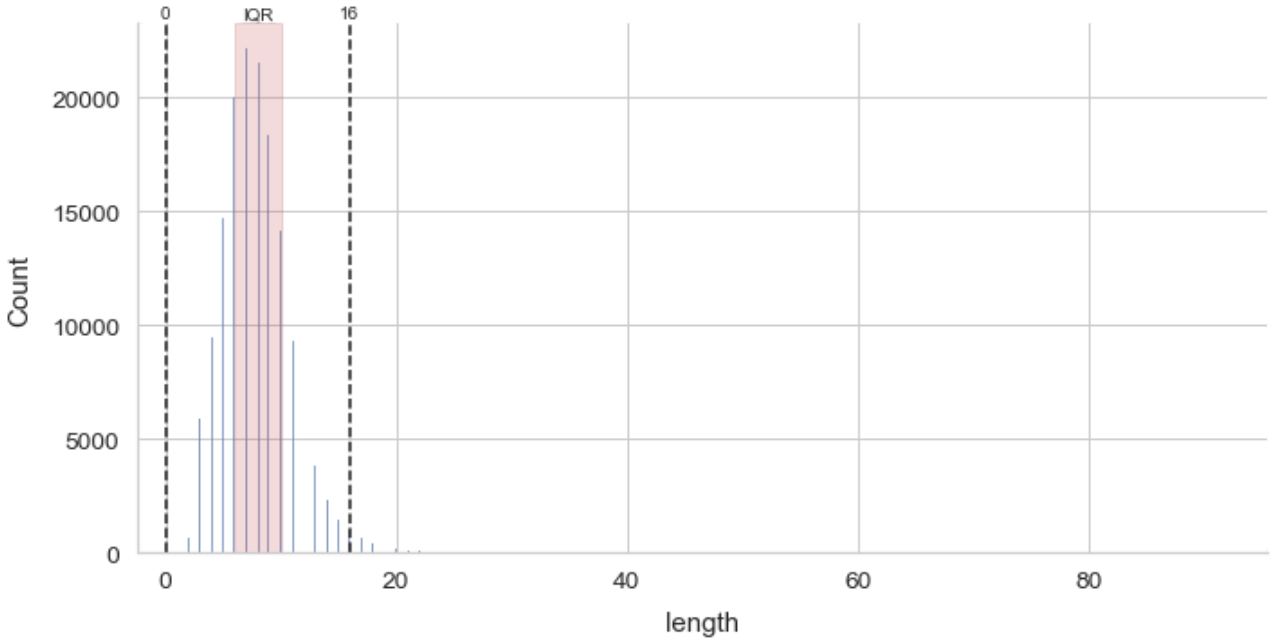
Out[30]:

	term	freq	length	uniq_ratio
149636	mosquitos	1	9	0.777778
33011	phenomenally	14	12	0.750000
66522	erected	3	7	0.714286
98600	redpolls	1	8	0.875000
55039	overstated	4	10	0.800000
13950	kernels	80	7	0.857143
13881	hardboard	80	9	0.666667
119047	pufflings	1	9	0.888889
54002	hofsommer	5	9	0.777778
106935	macivers	1	8	1.000000

Next I examine the length distribution to determine a good cutoff point for length outliers. The plot is annotated with the IQR and Tukey's fences (i.e. boxplot whiskers).

In [31]:
```
g = sns.displot(data=samp_vocab, x="length", kind="hist", aspect=2)
plotting.add_tukey_marks(samp_vocab["length"], g.axes[0, 0], num_format=".0f")
```

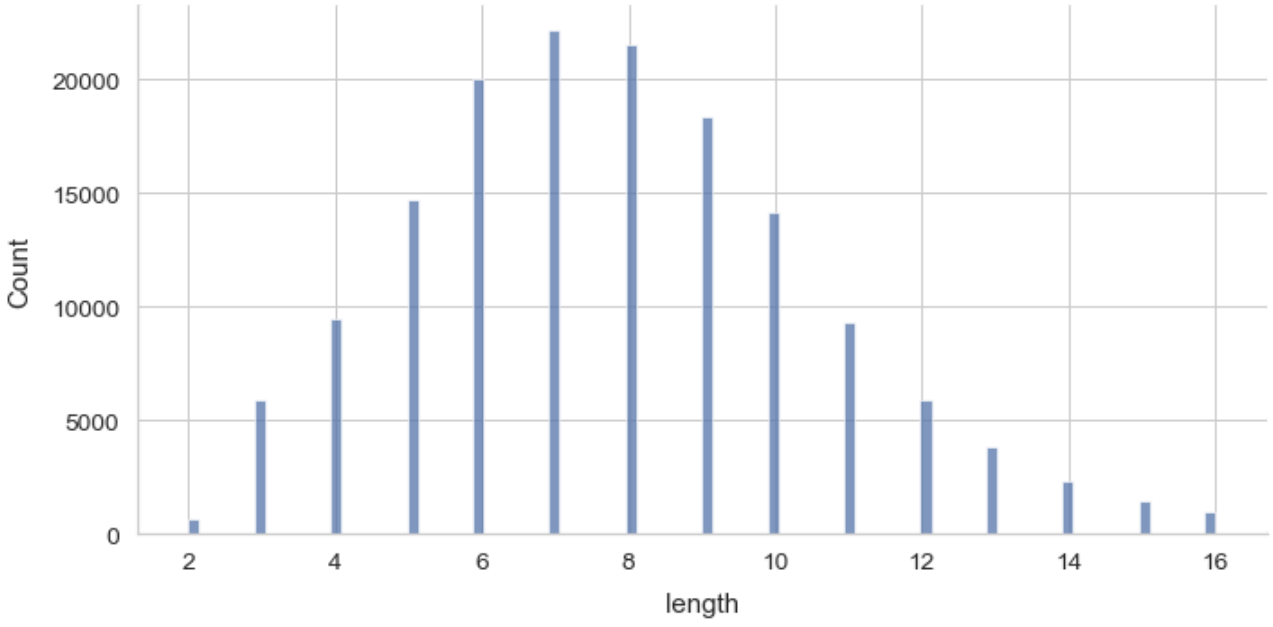
Out[31]: <AxesSubplot:xlabel='length', ylabel='Count'>



In [32]:
```
inlier_idx = outliers.tukey_trim(samp_vocab["length"]).index
samp_vocab = samp_vocab.loc[inlier_idx]
sns.displot(data=samp_vocab, x="length", kind="hist", aspect=2)
```

```
           n_trimmed  pct_trimmed
length         2,025            1
total_obs      2,025            1
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x1b95149f130>



16 seems like a natural cutoff for word length, and it's what Tukey's IQR proximity rule dictates. I rewrite my token pattern to match sequences between two and 16 characters (inclusive).

In [33]:
```
def my_tokenize_2(docs, n_jobs=None):
    pattern = re.compile(r"(?i)\b[a-z_]{2,16}\b")
    return lang.process_strings(docs, pattern.findall, n_jobs=n_jobs)
```

In [34]:
```
del my_tokenize_1
```

In [35]:
```
samp_vocab.loc[samp_vocab.term.str.count("_") > 2]
```

Out[35]:

	term	freq	length	uniq_ratio
152057	sheet_size_w_x_h	1	16	0.562500
108127	___description	1	15	0.733333
116762	___notes	1	9	0.666667
85590	_____	2	6	0.166667
39533	____	9	5	0.200000
85407	___	2	4	0.250000
51406	__	5	3	0.333333

I'm not interested in any of the underscores already in the data, only ones that I might add.

In [36]:
```
df["text"] = lang.strip_punct(df.loc[:, "text"], n_jobs=-1)
df["text"].str.contains("_").any()
```

Out[36]:
```
False
```

A related source of noise are strings with repetitive character (or word) sequences.

In [37]:
```
samp_vocab = build_sample_vocab(df.loc[:, "text"], 10 ** 5, tokenizer=my_tokenize_2)
samp_vocab.sort_values("uniq_ratio").head(10)
```

Out[37]:

	term	freq	length	uniq_ratio
97850	xxxxxxxxxxx	1	12	0.083333
102038	wooooooooooo	1	14	0.142857
133459	zzzzzzz	1	7	0.142857
131771	xxxxxxx	1	7	0.142857
138699	gaaaaaaaaaaa	1	13	0.153846
87573	cccccc	2	6	0.166667
91455	xxxxxx	2	6	0.166667
113454	ffffff	1	6	0.166667
56753	xxxxx	4	5	0.200000
90474	aaaaa	2	5	0.200000

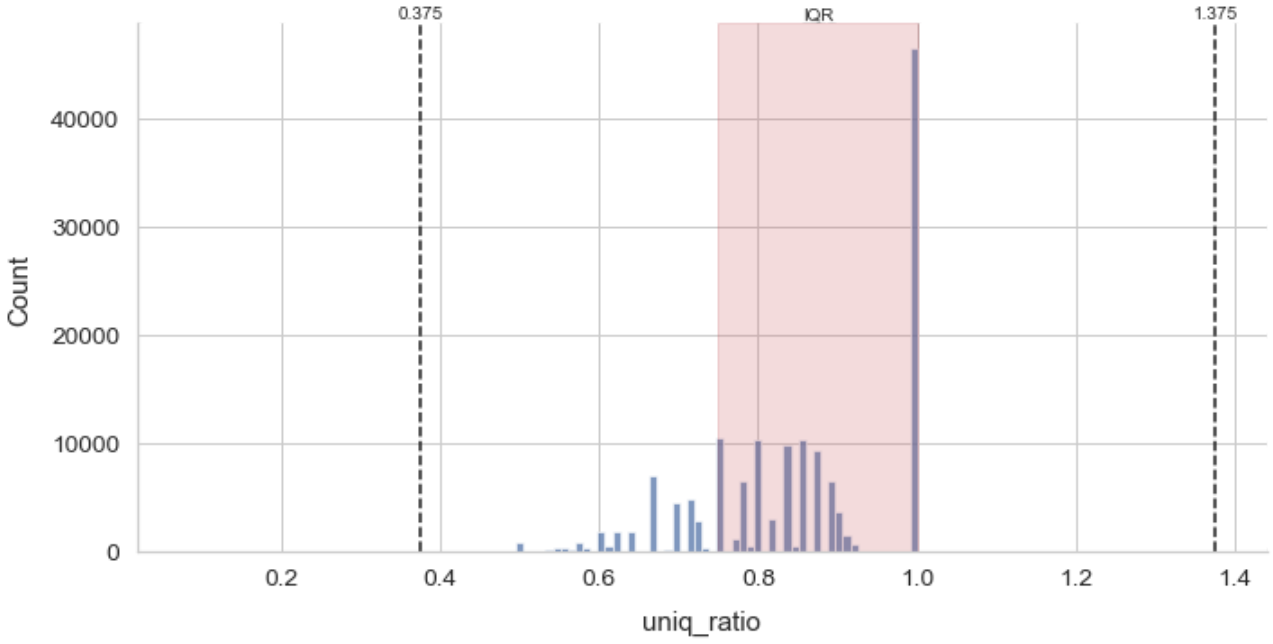
The above terms with repetitive sequences have low character uniqueness ratios, i.e. the ratio of character types

to character tokens. That's how I'll filter them out.

Note: long multi-word amalgamations also have low character uniqueness simply because of their length, but I've already weeded those out.

In [38]:
```
g = sns.displot(data=samp_vocab, x="uniq_ratio", kind="hist", aspect=2)
plotting.add_tukey_marks(samp_vocab["uniq_ratio"], g.axes[0, 0], num_format=".3f")
```

Out[38]:
```
<AxesSubplot:xlabel='uniq_ratio', ylabel='Count'>
```



The lower Tukey fence is located at 0.375. That seems like a reasonable cutoff.

In [39]:
```
# Tokenize the text
df["text"] = my_tokenize_2(df.loc[:, "text"], n_jobs=-1)
df["text"].head()
```

Out[39]:
```
asin
B000050AUH    [philips, sonicare, standard, brush, head, son...
B000050B62    [norelco, deluxe, reflex, action, cord, cordle...
B000050B63    [norelco, quadra, action, cord, cordless, rech...
B000050B64    [norelco, quadra, action, cord, cordless, rech...
B000050B65    [norelco, deluxe, quadra, action, cord, cordle...
Name: text, dtype: object
```

In [40]:
```
# functools.partial holds argument in place
uniq_char_thresh = partial(lang.uniq_char_thresh, thresh=0.375)

# Remove low uniqueness ratios
df["text"] = df.loc[:, "text"].map(uniq_char_thresh)
df["text"].head()
```

Out[40]:
```
asin
B000050AUH    [philips, sonicare, standard, brush, head, son...
B000050B62    [norelco, deluxe, reflex, action, cord, cordle...
B000050B63    [norelco, quadra, action, cord, cordless, rech...
B000050B64    [norelco, quadra, action, cord, cordless, rech...
B000050B65    [norelco, deluxe, quadra, action, cord, cordle...
Name: text, dtype: object
```

In [41]:
```
samp_vocab.loc[samp_vocab.uniq_ratio > 0.375].sort_values("uniq_ratio").head(10)
```

Out[41]:

	term	freq	length	uniq_ratio
76843	reengineering	2	13	0.384615
51917	totallytattoo	5	13	0.384615
109634	sleeplessness	1	13	0.384615
82505	nanna	2	5	0.400000
147866	bulbsbulbs	1	10	0.400000
144856	momentaneamente	1	15	0.400000
65398	tennesseee	3	10	0.400000
83745	aadaa	2	5	0.400000
142929	ssiii	1	5	0.400000
122878	ababa	1	5	0.400000

Looks better. If I raise the threshold more, I'll begin to filter out too many normal words.

Restricting to English

Next I run langdetect (a Google language detection algorithm) on the documents and select only the English product descriptions. I have nothing against the other languages, but there are not enough examples of them in the dataset to build a truly multilingual model. I don't want the model to associate Spanish words with 'amazon fashion', for example. That would generalize poorly.

This cell takes 30-40 minutes to run if `RUN_LANGDETECT` is set.

In [42]:
```
if RUN_LANGDETECT:
    # Wrapper of `langdetect.detect` with multiprocessing
    df["lang"] = lang.detect_lang(df.loc[:, "text"], seed=0, n_jobs=-1)
    df["lang"].to_json("data/langdetect.json")
else:
    df["lang"] = pd.read_json("data/langdetect.json", typ="series")

    # Run on missing
    missing = df["lang"].isna()
    if missing.any():
        df.loc[missing, "lang"] = lang.detect_lang(
            df.loc[missing, "text"].str.join(" "),
            seed=51,
            n_jobs=-1,
        )
        # Save new results
        df["lang"].to_json("data/langdetect.json")

df["lang"].value_counts()
```

Out[42]:
```
en    1066097
es        988
it        200
fr        200
pt        133
no        117
nl        116
de        112
da         84
id         56
```

```
ca          49
sv          39
af          29
ro          25
pl          22
so          19
tl          12
sl          12
sw          10
cs          10
hr          10
lt           9
et           7
cy           7
tr           5
hu           5
fi           3
sk           2
sq           2
lv           1
vi           1
Name: lang, dtype: int64
```

In [43]:
```
df = df.loc[df.lang == "en"]
df["lang"].value_counts()
```

Out[43]:
```
en    1066097
Name: lang, dtype: int64
```

In [44]:
```
len(df)
```

Out[44]:
```
1066097
```

Removing Stopwords

Before searching for collocations, I want to remove stopwords. As you can see below, there are quite a few in the vocabulary.

In [45]:
```
samp_vocab = build_sample_vocab(df.loc[:, "text"], 10 ** 4, sortby="freq")
samp_vocab.head(10)
```

Out[45]:

	term	freq	length	uniq_ratio
0	the	71611	3	1.0
1	and	65843	3	1.0
2	to	38229	2	1.0
3	of	35550	2	1.0
4	for	28322	3	1.0
5	with	26181	4	1.0
6	in	25225	2	1.0
7	is	21039	2	1.0
8	your	14172	4	1.0
9	this	12594	4	1.0

In [46]:

```
# Fetch stopwords
stop = lang.fetch_stopwords("nltk_english")
# Run through tokenizer
stop = {y for x in stop for y in my_tokenize_2(x)}
# Make sure there are no brand terms
stop = stop.difference(df["brand"])
pprint(stop, compact=True)
```

```
{'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and',
 'any', 'are', 'aren', 'as', 'be', 'because', 'been', 'before', 'being',
 'below', 'between', 'both', 'but', 'by', 'can', 'couldn', 'did', 'didn', 'do',
 'does', 'doesn', 'doing', 'don', 'down', 'during', 'each', 'few', 'for',
 'from', 'further', 'had', 'hadn', 'has', 'hasn', 'have', 'haven', 'having',
 'he', 'her', 'here', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'if',
 'in', 'into', 'is', 'isn', 'it', 'its', 'itself', 'just', 'll', 'ma', 'me',
 'mightn', 'more', 'most', 'mustn', 'my', 'myself', 'needn', 'no', 'nor', 'not',
 'now', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'our', 'ours',
 'ourselves', 'out', 'over', 'own', 're', 'same', 'shan', 'she', 'should',
 'shouldn', 'so', 'some', 'such', 'than', 'that', 'the', 'their', 'theirs',
 'them', 'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
 'through', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was', 'wasn',
 'we', 'were', 'weren', 'what', 'when', 'where', 'which', 'while', 'who',
 'whom', 'why', 'will', 'with', 'won', 'wouldn', 'you', 'your', 'yours',
 'yourself', 'yourselves'}
```

I add some more domain-specific stopwords.

In [47]:

```
more_stop = [
    "none",
    "inches",
    "shipping",
    "shipped",
    "shipper",
    "shipments",
    "weight",
    "pounds",
    "ounces",
    "asin",
    "item",
    "item_number",
    "model_number",
    "listed",
    "within",
    "apo",
    "fpo",
    "address",
    "addresses",
    "support",
    "eligible",
    "customer",
    "review",
    "reviews",
    "reviewer",
    "reviewed",
]
stop.update({y for x in more_stop for y in my_tokenize_2(x)})
pprint(stop, compact=True)
```

```
{'about', 'above', 'address', 'addresses', 'after', 'again', 'against', 'ain',
 'all', 'am', 'an', 'and', 'any', 'apo', 'are', 'aren', 'as', 'asin', 'be',
 'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by',
 'can', 'couldn', 'customer', 'did', 'didn', 'do', 'does', 'doesn', 'doing',
 'don', 'down', 'during', 'each', 'eligible', 'few', 'for', 'fpo', 'from',
 'further', 'had', 'hadn', 'has', 'hasn', 'have', 'haven', 'having', 'he',
 'her', 'here', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'if', 'in',
 'inches', 'into', 'is', 'isn', 'it', 'item', 'item_number', 'its', 'itself',
 'just', 'listed', 'll', 'ma', 'me', 'mightn', 'model_number', 'more', 'most',
```

```
'mustn', 'my', 'myself', 'needn', 'no', 'none', 'nor', 'not', 'now', 'of',
'off', 'on', 'once', 'only', 'or', 'other', 'ounces', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 'pounds', 're', 'review', 'reviewed',
'reviewer', 'reviews', 'same', 'shan', 'she', 'shipments', 'shipped',
'shipper', 'shipping', 'should', 'shouldn', 'so', 'some', 'such', 'support',
'than', 'that', 'the', 'their', 'theirs', 'them', 'themselves', 'then',
'there', 'these', 'they', 'this', 'those', 'through', 'to', 'too', 'under',
'until', 'up', 've', 'very', 'was', 'wasn', 'we', 'weight', 'were', 'weren',
'what', 'when', 'where', 'which', 'while', 'who', 'whom', 'why', 'will',
'with', 'within', 'won', 'wouldn', 'you', 'your', 'yours', 'yourself',
'yourselves'}
```

In [48]:
```
samp_vocab = build_sample_vocab(df["text"], 10 ** 4, sortby="freq")
samp_vocab.head(10)
```

Out[48]:

	term	freq	length	uniq_ratio
0	the	71611	3	1.0
1	and	65843	3	1.0
2	to	38229	2	1.0
3	of	35550	2	1.0
4	for	28322	3	1.0
5	with	26181	4	1.0
6	in	25225	2	1.0
7	is	21039	2	1.0
8	your	14172	4	1.0
9	this	12594	4	1.0

In [49]:
```
# functools.partial wraps the func and holds `stop` in place
remove_my_stop = partial(lang.remove_stopwords, stopwords=stop)

df = df.assign(text=df.loc[:, "text"].map(remove_my_stop))
df["text"].head()
```

Out[49]:
```
asin
B000050AUH    [philips, sonicare, standard, brush, head, son...
B000050B62    [norelco, deluxe, reflex, action, cord, cordle...
B000050B63    [norelco, quadra, action, cord, cordless, rech...
B000050B64    [norelco, quadra, action, cord, cordless, rech...
B000050B65    [norelco, deluxe, quadra, action, cord, cordle...
Name: text, dtype: object
```

Multi-word Expressions

I already have a lot of high quality brand phrases which I expect to be strong features for the model. I'll go ahead and prepare those.

In [50]:
```
brand_ng = df.loc[:, "brand"].drop_duplicates().str.split("_").map(tuple)
brand_ng = brand_ng.loc[brand_ng.map(len) > 1]
brand_ng
```

Out[50]:
```
asin
B000050AUH              (philips, sonicare)
B000050B6B               (philips, norelco)
```

```
B0000531WK                    (oreal, paris)
B0000532QF                  (tom, of, maine)
B000066SYD                     (aqua, fresh)
                           ...
B00008YGO2                    (mumbo, jumbo)
B000ASBKGA                    (cta, digital)
B0017IK8W4                    (deep, silver)
B001K7HUUK           (alawar, entertainment)
B00GQVGV9Q    (big, leap, studios, pvt, ltd)
Name: brand, Length: 6777, dtype: object
```

I search for collocations in the text in a stratified fashion, scanning the documents of each 'main_cat' category as a separate corpus. The goal is to find ngrams which are peculiar to each category. I use pointwise mutual information (PMI) to identify bigrams.

In [51]:

```
cat_bg = lang.stratified_ngrams(
    # Join tokens back together
    df.assign(text=df.loc[:, "text"].str.join(" ")),
    n=2,
    text="text",
    # Stratify by 'main_cat'
    cat="main_cat",
    tokenizer=my_tokenize_2,
    # Use pointwise mutual information
    metric="pmi",
    # Select top 75% of scores in each category
    select_best=0.75,
    # Filter out ultra-rare ngrams
    min_freq=100,
    n_jobs=-1,
)
cat_bg
```

Out[51]:

	bigram	score	main_cat
0	(year, warranty)	9.750503	all beauty
1	(tom, maine)	9.687209	all beauty
2	(old, spice)	9.514352	all beauty
3	(diamond, fx)	9.006641	all beauty
4	(pro, health)	8.985854	all beauty
...	...	...	...
114142	(two, player)	4.935113	video games
114143	(additional, features)	4.924453	video games
114144	(gives, players)	4.897825	video games
114145	(xbox, account)	4.892873	video games
114146	(ultimate, edition)	4.882772	video games

114147 rows × 3 columns

In [52]:

```
# Merge ngrams into single list
phrases = cat_bg.bigram.append(brand_ng).drop_duplicates().to_list()
display(len(phrases))
phrases[:10]
```

```
98443
```

Out[52]:
```
[('year', 'warranty'),
 ('tom', 'maine'),
 ('old', 'spice'),
 ('diamond', 'fx'),
 ('pro', 'health'),
 ('face', 'paint'),
 ('philips', 'sonicare'),
 ('philips', 'norelco'),
 ('brand', 'new'),
 ('high', 'quality')]
```

In [53]:
```
ngram_tokenize = nltk.MWETokenizer(phrases).tokenize
ngram_tokenize
```

Out[53]:
```
<bound method MWETokenizer.tokenize of <nltk.tokenize.mwe.MWETokenizer object at 0x000001B9988E87F0
>>
```

In [54]:
```
df["text"] = df.loc[:, "text"].map(ngram_tokenize)

df["text"].head()
```

Out[54]:
```
asin
B000050AUH    [philips_sonicare, standard, brush_head, sonic...
B000050B62    [norelco, deluxe, reflex, action, cord, cordle...
B000050B63    [norelco, quadra, action, cord, cordless, rech...
B000050B64    [norelco, quadra, action, cord, cordless, rech...
B000050B65    [norelco, deluxe, quadra, action, cord, cordle...
Name: text, dtype: object
```

Next, I add stopwords which I've seen in a vision of the future.

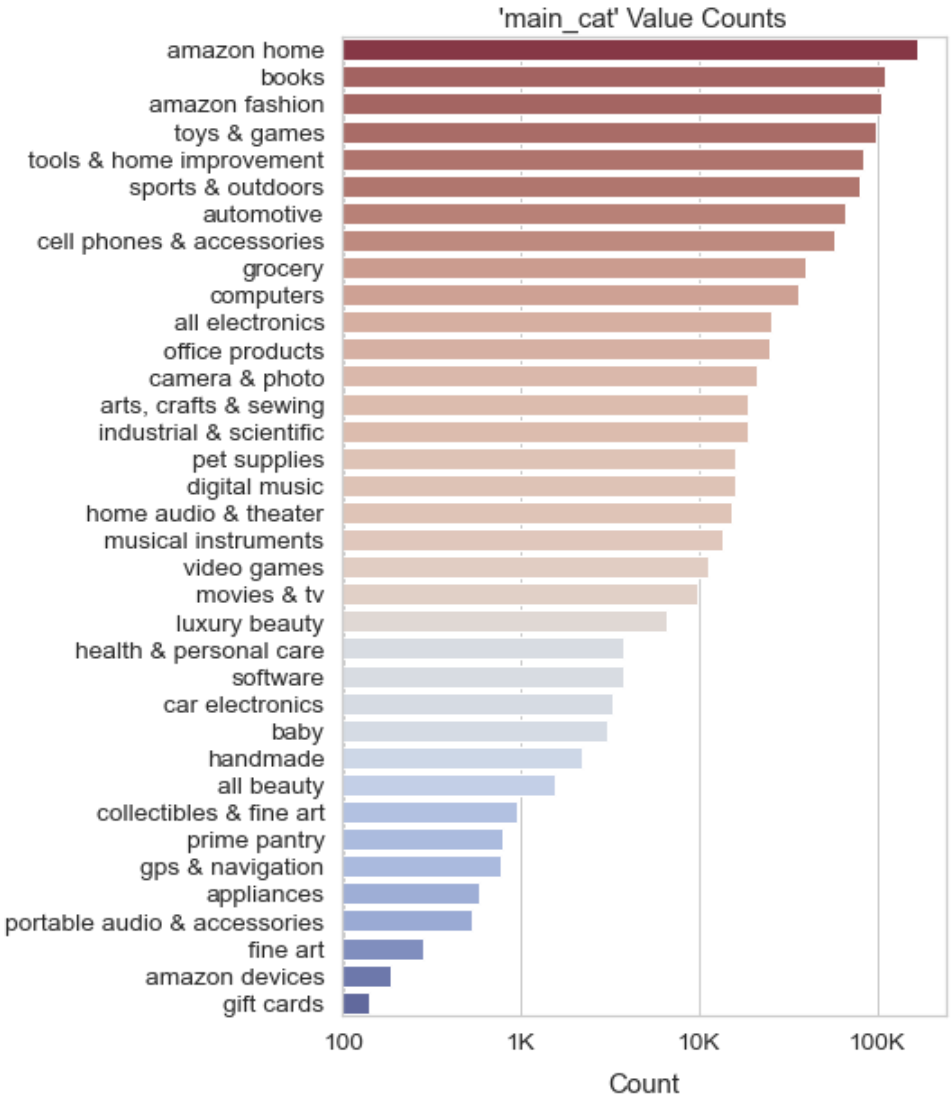
In [55]:
```
df["text"] = df.loc[:, "text"].str.join(" ")
df["text"].head()
```

Out[55]:
```
asin
B000050AUH    philips_sonicare standard brush_head sonic wav...
B000050B62    norelco deluxe reflex action cord cordless rec...
B000050B63    norelco quadra action cord cordless rechargeab...
B000050B64    norelco quadra action cord cordless rechargeab...
B000050B65    norelco deluxe quadra action cord cordless rec...
Name: text, dtype: object
```

In [56]:
```
if os.path.exists("data/model_data"):
    shutil.rmtree("data/model_data")

df.to_parquet(
    "data/model_data",
    engine="pyarrow",
    index=True,
    partition_cols="main_cat",
)
```

Before I move on to modeling, I plot the final class balance.

In [57]:
```
ax = plotting.plots.countplot(df.loc[:, "main_cat"], size=(6, 10), log_scale=True)
ax.xaxis.set_major_formatter(plotting.big_number_formatter())
ax.figure.savefig(
    "figures/class_bal.svg",
    bbox_inches="tight",
)
```



Modeling

Performing the Split

In [86]:
```
df = pd.read_parquet("data/model_data", engine="pyarrow")
df.head()
```

Out[86]:

asin	text	title	brand	lang	main_cat
B000050AUH	philips_sonicare standard brush_head sonic wav...	philips sonicare standard brush head	philips_sonicare	en	all beauty
B000050B62	norelco deluxe reflex action cord cordless rec...	norelco 5841xl deluxe reflex action cord/cordl...	norelco	en	all beauty
B000050B63	norelco quadra action cord cordless rechargeab...	norelco 6826xl quadra action cord/cordless rec...	norelco	en	all beauty
B000050B64	norelco quadra action cord cordless rechargeab...	norelco 6865xl quadra action cord/cordless rec...	norelco	en	all beauty

asin	text	title	brand	lang	main_cat
B000050B65	norelco deluxe quadra action cord cordless rec...	norelco 6885xl deluxe quadra action cord/cordl...	norelco	en	all beauty

In [87]:
```
X = df.loc[:, "text"]
y = df.loc[:, "main_cat"]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=49, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[87]:
```
((799572,), (266525,), (799572,), (266525,))
```

Next, I create the vectorizer. I have a taste for interpretability and transparency, which is why I've opted to use term-frequency vectorization (e.g. binary occurrence, raw term-frequency, or TF*IDF). For the baseline model, I turn off IDF weighting and normalization. Therefore, `tf` is initially set to extract raw term-frequency vectors from the text.

In [88]:
```
tf = TfidfVectorizer(
    lowercase=False,
    token_pattern=r"(?i)\b[a-z_]{2,16}\b",
    norm=None,
    use_idf=False,
)

tf
```

Out[88]:
```
TfidfVectorizer(lowercase=False, norm=None,
                token_pattern='(?i)\\b[a-z_]{2,16}\\b', use_idf=False)
```

I've opted to use Scikit-Learn's `SGDClassifier` because it's efficient on large datasets and works with several different loss functions. Essentially, it offers widely used linear classifiers with stochastic gradient descent optimization.

I could have also chosen a neural network classifier such as Scikit-Learn's ready-made multi-layer perceptron. However, neural networks take a long time to train and are the least interpretable algorithms on the market. For this project, given my limited computational resources and personal tastes, I've opted for a more traditional approach.

In [89]:
```
sgd = SGDClassifier(
    loss="hinge",
    class_weight="balanced",
    n_jobs=-1,
)

sgd
```

Out[89]:
```
SGDClassifier(class_weight='balanced', n_jobs=-1)
```

The default loss function 'hinge' is a linear support vector classifier. `SGDClassifier` also offers squared hinge, logistic regression, modified Huber, and several losses designed for regression.

Next, I create a pipeline containing just two steps: the vectorizer and the classifier.

In [90]:
```
pipe = Pipeline(
    [
        ("vec", tf),
        ("cls", sgd),
    ],
    verbose=True,
)
pipe
```

Out[90]:
```
Pipeline(steps=[('vec',
                 TfidfVectorizer(lowercase=False, norm=None,
                                 token_pattern='(?i)\\b[a-z_]{2,16}\\b',
                                 use_idf=False)),
                ('cls', SGDClassifier(class_weight='balanced', n_jobs=-1))],
         verbose=True)
```

The User Guide emphasizes that `SGDClassifier` is sensitive to feature scale. However, since word frequencies are already on the same scale, there is no need to include a scaler.

A related concern is vector normalization (sample scale). The User Guide states that the default learning rate formula (`learning_rate='optimal'`) assumes "that the norm of the training samples is approx. 1." Another reason to normalize the vectors is to minimize the effect of document length. Again, I've turned it off for the baseline, but I fully expect to turn it back on later.

Next I define some functions which I'll use throughout the modeling process.

In [91]:
```
def plot_confusion_matrix(
    estimator,
    X_test=X_test,
    y_test=y_test,
    dst=None,
    xticks_rotation="vertical",
    **kwargs,
):
    """Plots confusion matrix with accuracy score."""
    ax = plotting.confusion_matrix(
        estimator,
        X_test,
        y_test,
        xticks_rotation=xticks_rotation,
        **kwargs,
    )
    y_pred = estimator.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    bal_acc = balanced_accuracy_score(y_test, y_pred)
    ax.set_title(f"Accuracy: {acc:.2f}, Balanced Accuracy: {bal_acc:.2f}", pad=5)
    if isinstance(dst, str):
        plt.savefig(dst, bbox_inches="tight")
    return ax
```

In [92]:
```
def get_report(estimator, X_test=X_test, y_test=y_test):
    """Returns standard classification metrics as a DataFrame."""
    report = classification_report(y_test, estimator.predict(X_test), output_dict=True)
    report = pd.DataFrame(report).T
    report.loc["accuracy", "support"] = report["support"].max()
    return report
```

In [93]:
```
def extract_coef(
    pipeline,
```

```
    classifier="cls",
    vectorizer="vec",
):
    """Returns labeled model coefficients as a DataFrame."""
    columns = np.array(pipeline[vectorizer].get_feature_names())
    coef = pd.DataFrame(
        pipeline[classifier].coef_,
        index=pipeline[classifier].classes_,
        columns=columns,
    ).T
    return coef
```

Fitting the Baseline

Without further ado, it's time to fit the baseline model.

In [94]:
```
if "svm_1" in FIT_MODELS:
    # Train the model (slow)
    pipe.fit(X_train, y_train)
    joblib.dump(pipe, "models/svm_1.joblib", compress=True)

else:
    # Load the saved model (fast)
    pipe = joblib.load("models/svm_1.joblib")
pipe
```

Out[94]:
```
Pipeline(steps=[('vec',
                 TfidfVectorizer(lowercase=False, norm=None,
                                 token_pattern='(?i)\\b[a-z_]{2,16}\\b',
                                 use_idf=False)),
                ('cls', SGDClassifier(class_weight='balanced', n_jobs=-1))],
         verbose=True)
```

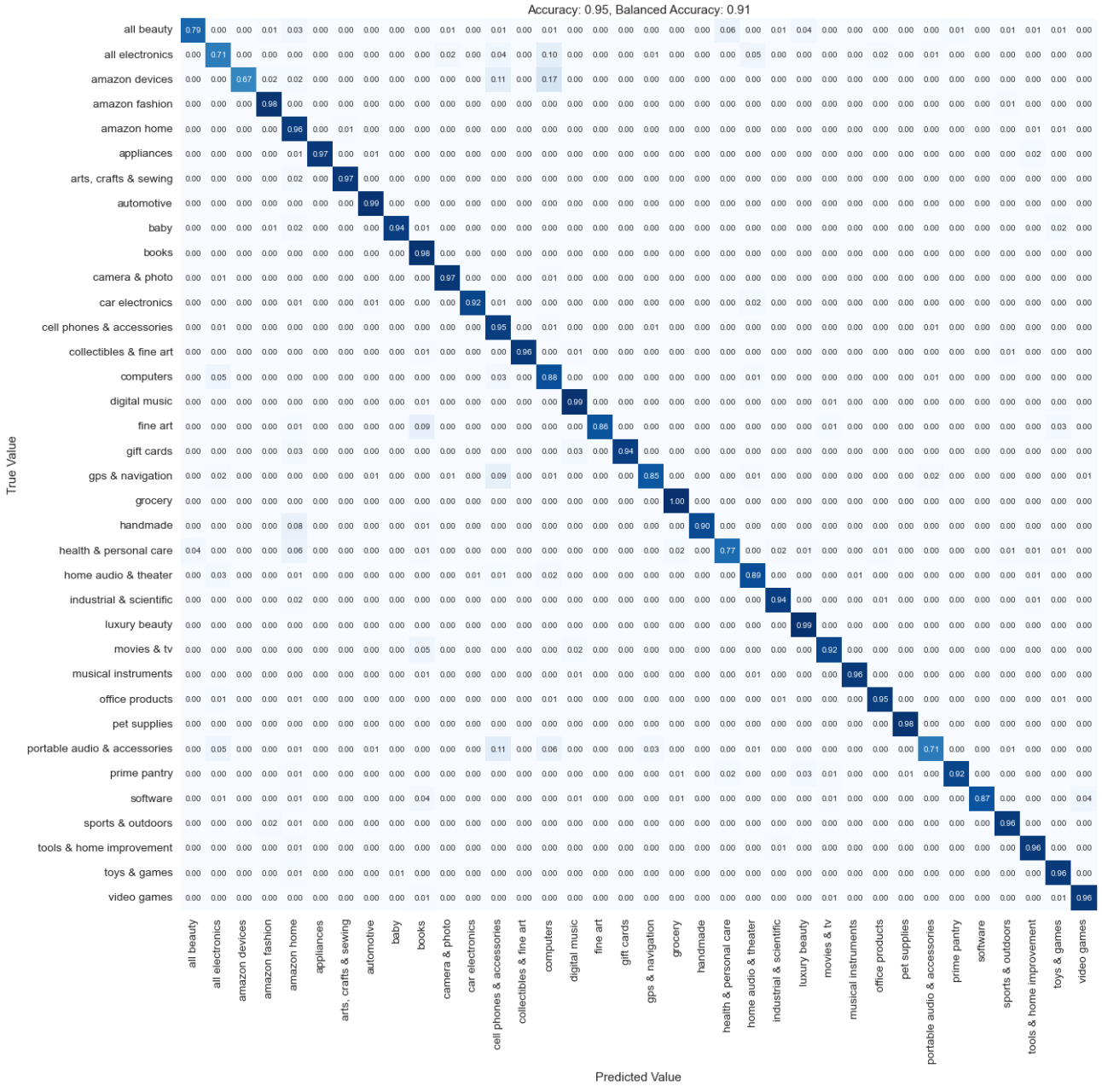
Since these models take a couple minutes to train, I always save them so they can be quickly reloaded.

In [95]:
```
len(pipe["vec"].get_feature_names())
```

Out[95]:
```
491581
```

490K features is a lot, but not bad considering the size of the dataset. I'm sure it would've been much larger if not for my extensive preprocessing efforts.

In [96]:
```
name = "svm_1"
if name in FIT_MODELS:
    ax = plotting.confusion_matrix(pipe, X_test, y_test, size=(20, 20))
    joblib.dump(ax.figure, f"figures/{name}.joblib")

else:
    fig = joblib.load(f"figures/{name}.joblib")

fig.savefig(f"figures/{name}.svg", bbox_inches="tight")
fig
```



Not bad for a baseline! Is it too good to be true, or have my painstaking efforts simply paid off? Let's look at the coefficients.

In [97]:

```
coef = extract_coef(pipe)
coef.sort_values("all electronics", ascending=False).head(10)
```

Out[97]:

	all beauty	all electronics	amazon devices	amazon fashion	amazon home	appliances	arts, crafts & sewing	automotive	baby	
riorand	-0.037515	1.799621	0.000000	-0.025009	-0.124012	-0.002779	0.000000	-0.046893	-0.003126	0.
kenable	0.000000	1.136051	0.000000	-0.014068	-0.009379	-0.002779	0.000000	-0.018757	-0.004689	0.
amscope	0.000000	1.104640	-0.011463	0.000000	-0.027095	0.000000	0.000000	-0.017194	0.000000	0.
silverstone	0.000000	1.072703	0.000000	0.004354	-0.001812	-0.002779	0.079546	-0.007815	0.000000	0.
javoedge	-0.012505	1.000720	0.487363	-0.003126	-0.010421	-0.002779	-0.006252	0.000000	0.000000	0.
thermaltake	-0.002779	1.000517	-0.005211	0.000000	-0.001042	-0.008337	0.000000	-0.068082	0.000000	-0.

	all beauty	all electronics	amazon devices	amazon fashion	amazon home	appliances	arts, crafts & sewing	automotive	baby	
ocz	0.000000	0.993562	-0.002084	0.000000	0.000000	-0.011116	0.000000	-0.010942	0.000000	0.
altec_lansing	0.000000	0.982081	-0.014590	0.000000	0.000000	-0.001389	0.000000	0.000000	0.000000	0.
trendnet	0.000000	0.939626	-0.009379	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
pny	0.000000	0.924110	-0.002084	0.000000	0.000000	0.000000	0.000000	-0.023446	0.000000	0.

10 rows × 36 columns

The coefficients look just as I expected. Brand terms are at the top, at least for 'all electronics'. While I did filter out only the top 1% of brands for each category, there are still nearly 10,000 brands in the dataset. Moreover, these brand names are recognizable and make sense.

Next, it's time to choose the loss function.

Selecting the Classifier

I try out all four of the classification loss functions: 'hinge', 'squared_hinge', 'log', and 'modified_huber'. I also tune the penalty and regularization strength, as well as the basic vectorizer settings.

In [98]:
```
sgd_grid = {
    "cls__loss": ["hinge", "squared_hinge", "log", "modified_huber"],
    "cls__penalty": ["l1", "l2"],
    "cls__alpha": sp.stats.loguniform(1e-8, 1.0),
    "vec__use_idf": [True, False],
    "vec__binary": [True, False],
    "vec__norm": ["l1", "l2", None],
}

sgd_grid
```

Out[98]:
```
{'cls__loss': ['hinge', 'squared_hinge', 'log', 'modified_huber'],
 'cls__penalty': ['l1', 'l2'],
 'cls__alpha': <scipy.stats._distn_infrastructure.rv_frozen at 0x1bfbe145f40>,
 'vec__use_idf': [True, False],
 'vec__binary': [True, False],
 'vec__norm': ['l1', 'l2', None]}
```

Given the size of the dataset, with `X_train` at around 800K samples, I opt for a randomized search with successive halving. It's essentially a survival-of-the-fittest contest between a set of randomly-selected parameter combinations. In the first round, the parameters are tested on 10K samples. With each successive round, the weak combinations are eliminated and the amount of data increases.

It's a more scalable approach than running an exhaustive search on the full `X_train`, though it still takes an hour or two. Note that I am optimizing **accuracy** while keeping an eye on "balanced accuracy" a.k.a. macro average (unweighted) recall. I'm mostly concerned with the overall ratio of correct classification, as long as the small classes are reasonably accommodated.

In [99]:
```
if "loss" in RUN_SWEEPS:
    gs = selection.sweep(
        pipe,
        sgd_grid,
        X=X_train,
```

```
        y=y_train,
        cv_dst="sweeps/loss_cv.joblib",
        kind="hrand",
        min_resources=10 ** 4,
        max_resources=X_train.shape[0],
        n_jobs=-1,
        factor=3,
    )
    loss_results = gs.cv_results_
else:
    loss_results = joblib.load("sweeps/loss_cv.joblib")

loss_results = selection.prune_cv(loss_results)
loss_results.head(10)
```

Out[99]:

	alpha	loss	penalty	binary	norm	use_idf	params	mean_fit_time	mean_score	rank_score
0	5.255038e-07	hinge	l2	True	l2	True	{'cls__alpha': 5.25503769545935e-07, 'cls__los...	241.589407	0.963798	1
1	5.255038e-07	hinge	l2	True	l2	True	{'cls__alpha': 5.25503769545935e-07, 'cls__los...	121.395599	0.953789	2
2	7.157636e-07	log	l2	False	l2	True	{'cls__alpha': 7.15763577656479e-07, 'cls__los...	102.892204	0.949059	3
3	1.605315e-08	log	l2	False	l1	True	{'cls__alpha': 1.6053147327650663e-08, 'cls__l...	99.634805	0.947215	4
4	7.157636e-07	log	l2	False	l2	True	{'cls__alpha': 7.15763577656479e-07, 'cls__los...	26.191196	0.935344	5
5	5.255038e-07	hinge	l2	True	l2	True	{'cls__alpha': 5.25503769545935e-07, 'cls__los...	26.031597	0.933922	6
6	1.605315e-08	log	l2	False	l1	True	{'cls__alpha': 1.6053147327650663e-08, 'cls__l...	28.755394	0.933267	7
7	3.082426e-06	hinge	l2	False	l2	False	{'cls__alpha': 3.0824261182174037e-06, 'cls__l...	24.616992	0.928700	8
8	7.857063e-06	hinge	l2	False	l2	False	{'cls__alpha': 7.857063428799315e-06, 'cls__lo...	23.931392	0.924689	9
9	4.430582e-08	log	l2	True	l1	True	{'cls__alpha': 4.4305817166390654e-08, 'cls__l...	27.540397	0.924622	10

The default hinge loss wins, with logistic regression in a close second. L2 penalty looks to have completely dominated L1, and the top-scoring 'alpha' values are on the order of 10^7 or 10^8. Unsurprisingly, the vector normalization was selected (L2) and IDF weighting to reduce the impact of common terms. Binary occurrence was selected over term-frequency, which I wouldn't have predicted.

For each document and each term, the vectorizer will mark whether the term occurs in that document. Then the vectorizer will apply IDF (inverse document frequency) weighting to the binary features, placing weight on rare

terms and withholding it from common ones. Then, finally, it will normalize the vectors to reduce the effect of document length (longer documents typically have a wider variety of terms, meaning more 1s in their vectors).

Fitting a Linear SVM

The next step is to set the new parameters and fit the second version of the model.

In [100…
```
loss_params = loss_results.loc[0, "params"]
display(loss_params)
pipe.set_params(**loss_params)
```

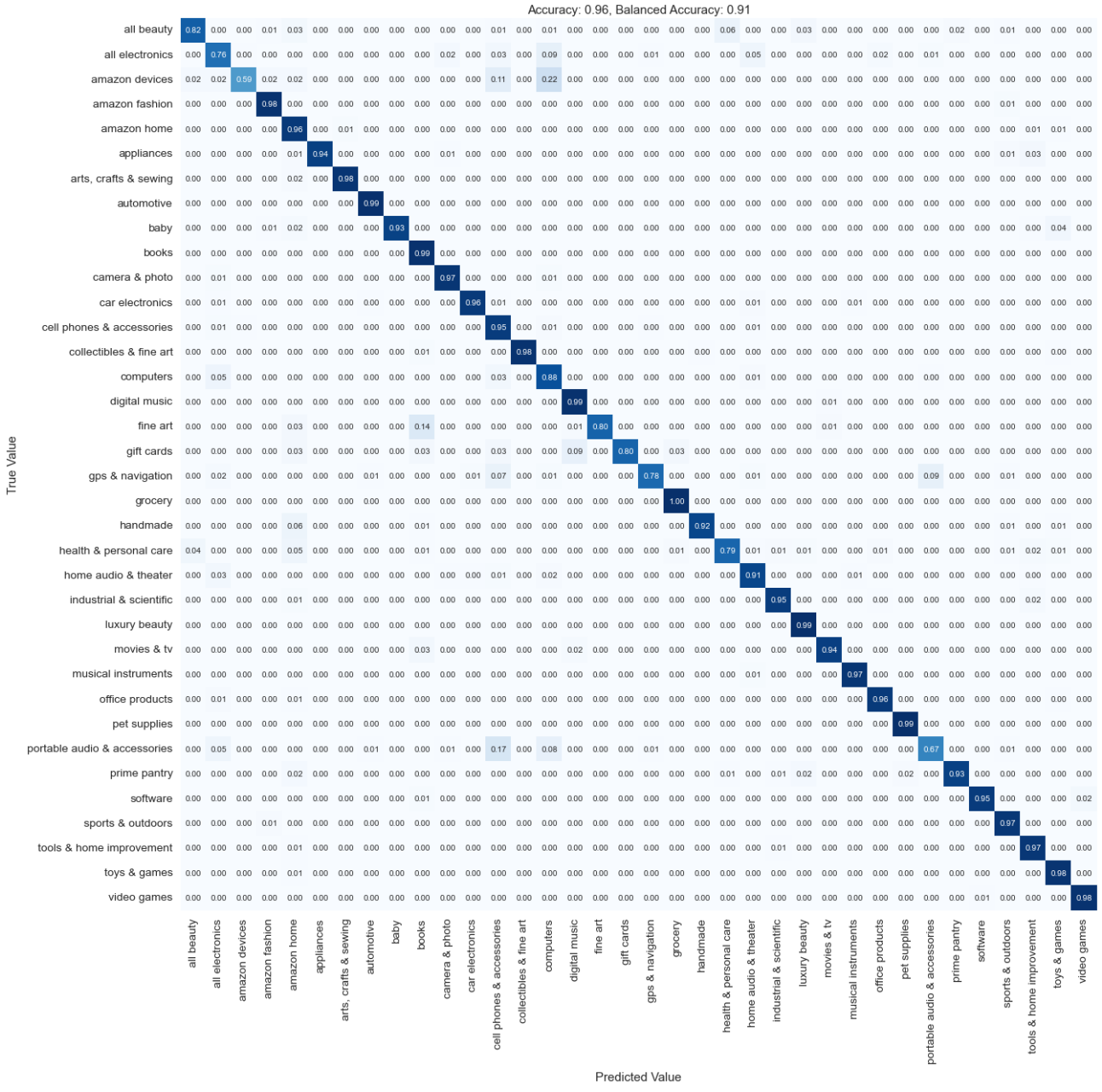
```
{'cls__alpha': 5.25503769545935e-07,
 'cls__loss': 'hinge',
 'cls__penalty': 'l2',
 'vec__binary': True,
 'vec__norm': 'l2',
 'vec__use_idf': True}
```

Out[100…
```
Pipeline(steps=[('vec',
                 TfidfVectorizer(binary=True, lowercase=False,
                                 token_pattern='(?i)\\b[a-z_]{2,16}\\b')),
                ('cls',
                 SGDClassifier(alpha=5.25503769545935e-07,
                               class_weight='balanced', n_jobs=-1))],
         verbose=True)
```

In [101…
```
if "svm_2" in FIT_MODELS:
    pipe.fit(X_train, y_train)
    joblib.dump(pipe, "models/svm_2.joblib", compress=True)

else:
    pipe = joblib.load("models/svm_2.joblib")
pipe
```

Out[101…
```
Pipeline(steps=[('vec',
                 TfidfVectorizer(binary=True, lowercase=False,
                                 token_pattern='(?i)\\b[a-z_]{2,16}\\b')),
                ('cls',
                 SGDClassifier(alpha=5.25503769545935e-07,
                               class_weight='balanced', n_jobs=-1))],
         verbose=True)
```

In [102…
```
name = "svm_2"
if name in FIT_MODELS:
    ax = plotting.confusion_matrix(pipe, X_test, y_test, size=(20, 20))
    joblib.dump(ax.figure, f"figures/{name}.joblib")

else:
    fig = joblib.load(f"figures/{name}.joblib")

fig.savefig(f"figures/{name}.svg", bbox_inches="tight")
fig
```



It looks like 'amazon devices' and 'portable audio & accessories' have the lowest recall. Amazon doesn't need this classifier for it's own proprietary devices, but 'portable audio' could be better. It's confused with 'cell phones & accessories' about 14% of the time.

Anyway, my focus is on overall accuracy, and I'm not interested in boosting the recall of small classes at the expense of large ones.

In [103…

```
coef = extract_coef(pipe)
coef.sort_values("video games", ascending=False).head(10)
```

Out[103…

	all beauty	all electronics	amazon devices	amazon fashion	amazon home	appliances	arts, crafts & sewing	automotive	baby
cta_digital	0.000000	-1.544874	-0.106037	-0.182259	-0.785255	0.000000	-0.062472	-0.516518	-0.114832
activision	0.000000	-0.412954	-0.039900	-0.124945	-0.227798	-0.023712	0.000000	-0.226012	-0.055396
electronic_arts	0.000000	-0.249129	0.000000	-0.205511	-0.696196	0.000000	0.000000	-0.430847	-0.025292

	all beauty	all electronics	amazon devices	amazon fashion	amazon home	appliances	arts, crafts & sewing	automotive	baby	
mad_catz	0.000000	-2.100931	-0.063326	0.000000	-0.146762	-0.046801	0.000000	-0.133605	-0.062200	
ubisoft	0.000000	-0.174991	-0.051661	-0.238391	-0.199343	0.000000	-0.060136	-0.318520	-0.039198	-
atari	0.000000	-0.258573	0.000000	-0.086140	-0.041062	0.000000	-0.055139	-0.158983	-0.046925	-
nintendo	-0.435857	-1.198789	-1.209829	-0.462205	-0.494124	-0.049790	-0.238836	-0.382382	-0.124717	-
thq	-0.087708	-0.178751	-0.017484	0.000000	0.000000	0.000000	0.000000	-0.347560	0.000000	-
sega	0.000000	-0.473691	0.000000	-0.069936	-0.068049	-0.047048	-0.171130	0.150439	-0.044313	-
sony	-0.652183	0.945628	-1.306150	-0.349223	-1.500289	-0.142507	-0.244259	-1.315657	-0.464903	-

10 rows × 36 columns

Another successful coefficient sanity check. Nearly all of the top features for video games are recognizable brand names.

Optimizing the Learning Rate

Next I'll optimize the learning rate for stochastic gradient descent. The default is 'optimal', which is a function of α, the current time step t, and a heuristic value t_0.

The other options I'll try are 'constant' and 'adaptive'. As the name suggests, 'constant' is just the constant eta_0, which is passed as a hyperparameter. For 'adaptive', the initial learning rate is eta_0, but when the stopping criterion is reached, the learning rate is divided by 5 and the descent continues (defying the gods). The descent doesn't stop until the learning rate goes below 10^{-6}.

In [104…
```
learn_grid = [
    {
        "learning_rate": ["constant", "adaptive"],
        "eta0": sp.stats.loguniform(1e-4, 1e4),
        "average": [True, False],
    },
    {
        "learning_rate": ["optimal"],
        "average": [True, False],
    },
]
learn_grid
```

Out[104…
```
[{'learning_rate': ['constant', 'adaptive'],
  'eta0': <scipy.stats._distn_infrastructure.rv_frozen at 0x1bf3c8a4c40>,
  'average': [True, False]},
 {'learning_rate': ['optimal'], 'average': [True, False]}]
```

In [105…
```
if "learn" in RUN_SWEEPS:
    gs = selection.sweep(
        pipe,
        learn_grid,
        X=X_train,
        y=y_train,
        cv_dst="sweeps/learn_cv.joblib",
        kind="hrand",
        add_prefix="cls__",
```

```
        min_resources=10 ** 4,
        max_resources=X_train.shape[0],
        n_jobs=-1,
        factor=3,
    )
    learn_results = gs.cv_results_
else:
    learn_results = joblib.load("sweeps/learn_cv.joblib")

learn_results = selection.prune_cv(learn_results)
learn_results.head(10)
```

Out[105…

	average	eta0	learning_rate	params	mean_fit_time	mean_score	rank_score
0	True	0.841666	adaptive	{'cls__average': True, 'cls__eta0': 0.84166636...	2718.412539	0.964341	1
1	True	0.841666	adaptive	{'cls__average': True, 'cls__eta0': 0.84166636...	971.841199	0.954659	2
2	False	1.656650	adaptive	{'cls__average': False, 'cls__eta0': 1.6566501...	362.309605	0.954233	3
3	False	0.857235	constant	{'cls__average': False, 'cls__eta0': 0.8572352...	85.645006	0.952689	4
4	True	0.841666	adaptive	{'cls__average': True, 'cls__eta0': 0.84166636...	289.311396	0.938111	5
5	False	1.656650	adaptive	{'cls__average': False, 'cls__eta0': 1.6566501...	117.517600	0.937444	6
6	False	0.857235	constant	{'cls__average': False, 'cls__eta0': 0.8572352...	36.053796	0.937389	7
7	False	1.925135	adaptive	{'cls__average': False, 'cls__eta0': 1.9251349...	116.465399	0.936867	8
8	False	0.086955	constant	{'cls__average': False, 'cls__eta0': 0.0869545...	26.184200	0.935878	9
9	False	0.081826	constant	{'cls__average': False, 'cls__eta0': 0.0818255...	87.510197	0.935878	9

Looks like 'adaptive' wins, and with an epic fit time. The long fit time makes sense given the unyielding nature of the algorithm.

Fitting the Final Model

Time to fit the third and final version of the model, now with 'adaptive' learning rate.

In [106…

```
learn_params = learn_results.loc[0, "params"]
display(learn_params)
pipe.set_params(**learn_params)
```

```
{'cls__average': True,
 'cls__eta0': 0.8416663608926325,
 'cls__learning_rate': 'adaptive'}
```

Out[106…

```
Pipeline(steps=[('vec',
                 TfidfVectorizer(binary=True, lowercase=False,
                                 token_pattern='(?i)\\b[a-z_]{2,16}\\b')),
                ('cls',
                 SGDClassifier(alpha=5.25503769545935e-07, average=True,
                               class_weight='balanced', eta0=0.8416663608926325,
                               learning_rate='adaptive', n_jobs=-1))],
         verbose=True)
```

In [107…

```
if "svm_3" in FIT_MODELS:
    pipe.fit(X_train, y_train)

    joblib.dump(pipe, "models/svm_3.joblib", compress=True)

else:
    pipe = joblib.load("models/svm_3.joblib")
pipe
```

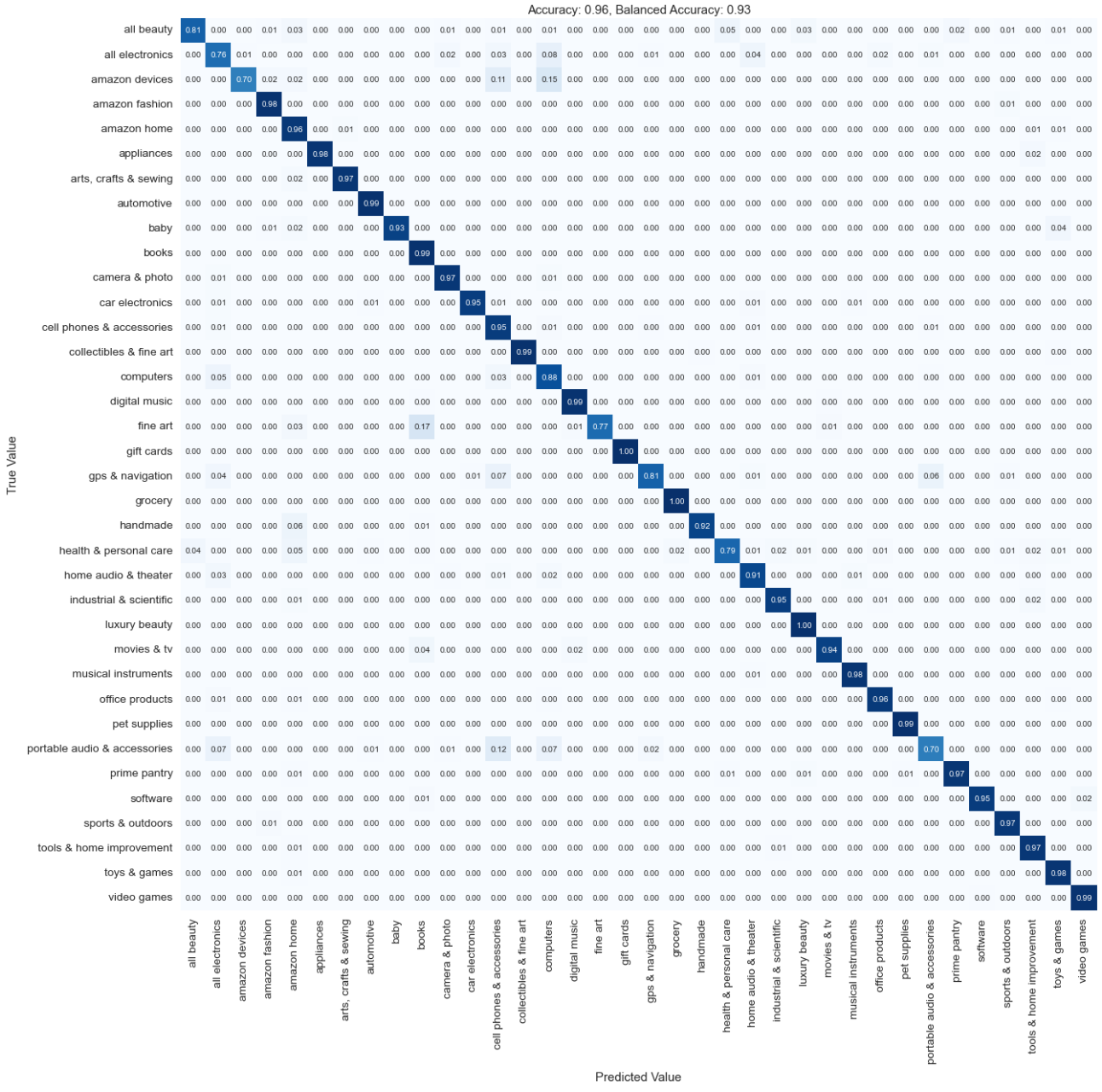
Out[107…

```
Pipeline(steps=[('vec',
                 TfidfVectorizer(binary=True, lowercase=False,
                                 token_pattern='(?i)\\b[a-z_]{2,16}\\b')),
                ('cls',
                 SGDClassifier(alpha=5.25503769545935e-07, average=True,
                               class_weight='balanced', eta0=0.8416663608926325,
                               learning_rate='adaptive', n_jobs=-1))],
         verbose=True)
```

That took ~10 minutes to train, which is about 5x a long as the 'optimal' learning rate.

In [108…

```
name = "svm_3"
if name in FIT_MODELS:
    ax = plotting.confusion_matrix(pipe, X_test, y_test, size=(20, 20))
    joblib.dump(ax.figure, f"figures/{name}.joblib")

else:
    fig = joblib.load(f"figures/{name}.joblib")

fig.savefig(f"figures/{name}.svg", bbox_inches="tight")
fig
```



It's still sitting at 0.96 accuracy, but the balanced accuracy improved slightly. 0.96 accuracy is already quite a satisfactory number, so I'll move on to interpretation.

Interpreting the Results

The next step is to take a closer look at the final model's coefficients. I already know that brand terms are critically important, but I'd like to see which brands are associated with which categories. I'm also interested in discovering what non-brand terms made it to the top.

In [109…

```
coef = extract_coef(pipe)
coef.columns = coef.columns.str.title()
coef.sort_values("Amazon Fashion", ascending=False).head(20)
```

Out[109…

	All Beauty	All Electronics	Amazon Devices	Amazon Fashion	Amazon Home	Appliances	Arts, Crafts & Sewing	Automotive	Baby

	All Beauty	All Electronics	Amazon Devices	Amazon Fashion	Amazon Home	Appliances	Arts, Crafts & Sewing	Automotive	Baby
port_authority	-0.212596	0.009663	-0.011277	6.053034	-0.222141	0.000000	-0.101590	-0.165922	-0.073788
fun_world	-0.185282	-0.013629	0.000000	5.044107	-1.082475	0.000000	-0.023725	-0.035658	-0.291685
sport_tek	0.000000	-0.014721	0.000000	5.021977	-0.056542	0.000000	-0.100879	-0.066158	0.000000
synthetic_sole	-0.223528	-0.073502	-0.074717	4.795533	-0.802506	0.000000	-0.190685	-0.301803	-0.290958
sk_hat_shop	-0.175141	0.000000	-0.022411	4.739004	-0.417239	-0.014362	-0.148723	-0.116165	-0.120623
fun_costumes	-0.058061	0.000000	-0.061565	4.477906	0.000000	0.000000	-0.126520	0.000000	-0.392929
rubber_sole	-0.356188	-0.217899	-0.155645	4.268081	-0.980572	-0.008216	0.026077	-0.219809	-0.405376
clothing_shoes	-0.403733	-0.109396	-0.015496	4.090943	-1.018837	0.000000	-0.794035	-0.057783	-0.440984
peach_couture	-0.199304	-0.130758	-0.043984	4.051408	-1.434661	0.000000	-0.243469	-0.038703	-0.161027
sole_heel	-0.332170	-0.009918	-0.123003	4.029316	-0.920196	-0.011871	-0.152868	-0.032036	-0.328577
greatlookz	-0.059315	0.000000	-0.004975	3.888560	-1.540568	0.000000	-0.400107	0.000000	-0.167027
kilofly	-0.393476	-0.246614	0.000000	3.842974	-0.621938	0.000000	-0.180492	-0.057149	-1.109661
damara_womens	-1.168214	-0.163282	-0.122791	3.836949	-0.436998	-0.015476	-0.091956	0.000000	-0.608669
tobeinstyle	-0.123597	0.000000	0.000000	3.775642	-0.428391	0.000000	0.000000	0.000000	-0.216726
michael_kors	-0.107783	-0.276432	-0.057959	3.753046	-0.391496	-0.014539	-0.371675	-0.052787	-0.214609
avalaya	-0.217298	-0.057528	-0.012006	3.741300	-1.357068	0.000000	-0.862267	-0.026775	-0.021855
harley_davidson	-0.279419	-0.477372	-0.074178	3.585215	1.342547	-0.042265	-0.423487	1.455907	-0.120810
watch	0.130324	0.028137	-1.795667	3.489432	-0.676668	-0.437934	-0.012770	-1.256815	-0.048455
casio	-0.125100	3.786088	-0.066038	3.481366	-0.761622	-0.114066	-0.115944	-0.085454	-0.043059
pewter_pendant	-0.348377	-0.066640	-0.017468	3.445926	-1.039632	0.000000	-0.741076	-0.020912	0.000000

20 rows × 36 columns

I definitely recognize some of these fashion brands, like Harley Davidson.

Since there are 36 categories and ~490K terms, it's difficult to get a view of the big picture. In order to get a birds-eye-view, I'm going to plot a small wordcloud for each of the top 9 categories, measured by F_1-score. Then I'll do the same for the bottom 9 categories.

I prepare some colormaps in the following cell.

In [110…

```
rng = np.random.default_rng(1594)

cmaps = [
    "Purples",
    "Blues",
    "Greens",
    "Oranges",
    "Reds",
    "YlOrBr",
    "YlOrRd",
    "OrRd",
    "PuRd",
    "RdPu",
```

```
    "BuPu",
    "GnBu",
    "PuBu",
    "YlGnBu",
    "PuBuGn",
    "BuGn",
    "YlGn",
]
rng.shuffle(cmaps)
cmaps
```

Out[110…
```
['Purples',
 'Greens',
 'BuGn',
 'YlGn',
 'GnBu',
 'RdPu',
 'YlOrRd',
 'BuPu',
 'PuBu',
 'PuRd',
 'Reds',
 'YlGnBu',
 'Oranges',
 'Blues',
 'PuBuGn',
 'OrRd',
 'YlOrBr']
```

Next, I get the scores for each category.

In [145…
```
report = get_report(pipe)
report.sort_values("f1-score", ascending=False).head()
```

Out[145…

	precision	recall	f1-score	support
grocery	0.998069	0.997764	0.997916	9840.0
books	0.990440	0.988471	0.989454	27669.0
automotive	0.986435	0.990697	0.988562	16662.0
luxury beauty	0.979798	0.995773	0.987721	1656.0
pet supplies	0.978587	0.993503	0.985989	4002.0

In [149…
```
top_f1 = report["f1-score"].nlargest(9).index.str.title()

fig = plotting.wordcloud(
    coef.loc[:, top_f1],
    cmap=rng.choice(cmaps, 9).tolist(),
)
fig.suptitle("Highest $F_{1}$-Scores", y=1.04, fontsize=16)
fig.savefig("figures/coef_top_f1.svg", bbox_inches="tight")
```

There are a lot of brand terms at the forefront, as I expected. However, There is also some category-specific fine print that shows up, especially in Grocery and Books. Legalistic phrases like "evaluated_fda", "fda_intended, "copyright_reed", and "rights_reserved" are among the top coefficients for these categories. This is probably because boilerplate category-related legalistic text appears repeatedly throughout the category, but does not appear in other categories.

In [147…

```
bottom_f1 = report["f1-score"].nsmallest(9).index.str.title()

fig = plotting.wordcloud(
    coef.loc[:, bottom_f1],
    cmap=rng.choice(cmaps, 9).tolist(),
)
fig.suptitle("Lowest $F_{1}$-Scores", y=1.04, fontsize=16)
fig.savefig("figures/coef_bot_f1.svg", bbox_inches="tight")
```

Brand names are on top across the board for the categories with the lowest F_1-scores. The classifier still does pretty well for these categories, as evidenced by the highly recognizable brand names.

Conclusion

I developed a highly accurate 36-class classifier for Amazon products using tried-and-true machine learning methods. One reason the model attained such a high score is that I selected a very high quality dataset. Amazon product data is both plentiful and well-labeled. The human-given category labels are highly accurate. Why? Because on Amazon, people's livelihoods are on the line (including Amazon's). It matters to sellers how they classify their products, and it matters to you too. Classification matters to businesses because it matters to customers—it can determine whether a customer buys a product or never even hears about it.

If you're looking to classify products with NLP, lead with the brand terms.

Brand terms ranked high in nearly every category. One could build a decent model with *only* brand terms, though I wouldn't recommend going that far. Even if you wanted an image-based classifier, brands are the first place I'd start.

Don’t ignore boilerplate legalistic text, because sometimes it’s category-specific.

In fact, I recommend you gather up all the legalistic caveats and copyright statements you can get. This text is sometimes very distinctive of its category.

Use the model to study your competitors and scope out new suppliers.

This model can be used to analyze other business' inventories, including those of competitors. Discover new products and suppliers by directly comparing their inventories to yours under your classification scheme. See how their classification differs from yours by examining how the categories line up.

Looking Forward

- Gather data on brands concerning their relationships and parent companies.
  - Try to expand the model's coverage to more obscure brands.
- Develop a workflow to create specialized subcategory models for each major category.
  - These will be **multilabel** classification models.
- Create a dashboard to demonstrate the accuracy and rich interpretability of the model.
- Obtain a new, unseen dataset to test the model's generalizability.

In [ ]: