Appendix A: API

ADS 509: Final Project Code

Team 4

Zachariah Freitas and Brianne Bell

In [1]:
```
# Loading necessary libraries
import pandas as pd
import numpy as np
import os
import re
import random
import time
import datetime

import nltk
from collections import Counter, defaultdict
from nltk.corpus import stopwords
from string import punctuation
from tqdm import tqdm
import sqlite3

# from https://github.com/soumik12345/multi-label-text-classification/blob/master/arxi
import arxiv
```

In [2]:
```
# doing similar set up with setting up keywords to focus on
## Alternative keywords can be used to attempt better model performance or for differe
query_keywords = [
    "\"representation learning\"",
    "\"image generation\"",
    "\"object detection\"",
    "\"transformers\"",
    "\"image segmentation\"",
    "\"natural language\"",
    "\"graph\"",
    "\"colorization\"",
    "\"depth estimation\"",
    "\"point cloud\"",
    "\"structured data\"",
    "\"reinforcement learning\"",
    "\"attention\"",
    "\"tabular\"",
    "\"unsupervised learning\"",
    "\"semi-supervised learning\"",
    "\"explainable\"",
    "\"time series\"",
    "\"molecule\"",
    "\"physics\"",
    "\"graphics\""
]
```

In [3]:
```
# https://github.com/soumik12345/multi-label-text-classification/blob/master/arxiv_scr

# We are pulling just the terms (topic), titles, and abstracts of articles
    ## with a limit of 20k for each to save time and space
# other queries can be pulled if needed/wanted but we are focusing on titles and abstr

client = arxiv.Client(num_retries=20, page_size=500)

def query_with_keywords(query):
    search = arxiv.Search(
        query=query,
        max_results=20000,
        sort_by=arxiv.SortCriterion.LastUpdatedDate
    )
    terms = []
    titles = []
    abstracts = []
    for res in tqdm(client.results(search), desc=query):
        if res.primary_category in ["cs.CV", "stat.ML", "cs.LG"]:
            terms.append(res.categories)
            titles.append(res.title)
            abstracts.append(res.summary)
    return terms, titles, abstracts
```

In [4]:
```
# setting up save file
# if not os.path.isdir("arxiv_data") :
#     os.mkdir("arxiv_data")
```

In [5]:
```
# setting up for pull
all_titles = []
all_summaries = []
all_terms = []

# timing:
start_time = datetime.datetime.now()

# pulling
for query in query_keywords:
    terms, titles, abstracts = query_with_keywords(query)
    all_titles.extend(titles)
    all_summaries.extend(abstracts)
    all_terms.extend(terms)

# seeing how long ^that took:
end_time = datetime.datetime.now()
print(end_time - start_time)
```

```
"representation learning": 6118it [01:41, 60.09it/s]
"image generation": 1978it [00:30, 64.01it/s]
"object detection": 6536it [02:01, 53.69it/s]
"transformers": 20000it [06:55, 48.10it/s]
"image segmentation": 2890it [00:43, 66.93it/s]
"natural language": 13021it [03:36, 60.18it/s]
"graph": 20000it [05:39, 58.89it/s]
"colorization": 20000it [05:37, 59.20it/s]
"depth estimation": 1218it [00:20, 60.71it/s]
"point cloud": 4308it [01:30, 47.41it/s]
"structured data": 1915it [00:35, 54.30it/s]
"reinforcement learning": 16211it [04:36, 58.58it/s]
"attention": 20000it [05:48, 57.44it/s]
"tabular": 1382it [00:21, 63.47it/s]
"unsupervised learning": 2763it [00:41, 66.14it/s]
"semi-supervised learning": 0it [00:03, ?it/s]
"explainable": 20000it [06:17, 52.97it/s]
"time series": 15302it [04:17, 59.39it/s]
"molecule": 20000it [05:29, 60.67it/s]
"physics": 20000it [08:39, 38.50it/s]
"graphics": 15861it [04:34, 57.71it/s]
1:10:04.621341
```

Interpretting the scraping results:

- "representation learning": went through 6118 iterations at 60.09it/s.
- "image generation": went through 1978 iterations at 64.01it/s]
- "object detection": went through 6536 iterations at 53.69it/s.
- "transformers": went through 20000 iterations at 48.10it/s.
- "image segmentation": went through 2890 iterations at 66.93it/s.
- "natural language": went through 13021 iterations at 60.18it/s.
- "graph": went through 20000 iterations at 58.89it/s.
- "colorization": went through 20000 iterations at 59.20it/s.
- "depth estimation": went through 1218 iterations at 60.71it/s.
- "point cloud": went through 4308 iterations at 47.41it/s.
- "structured data": went through 1915 iterations at 54.30it/s.
- "reinforcement learning": went through 16211 iterations at 58.58it/s.
- "attention": went through 20000 iterations at 57.44it/s.
- "tabular": went through 1382 iterations at 63.47it/s.
- "unsupervised learning": went through 2763 iterations at 66.14it/s.
- "semi-supervised learning": went through 0 iterations at ?it/s.
- "explainable": went through 20000 iterations at 52.97it/s.
- "time series": went through 15302 iterations at 59.39it/s.
- "molecule": went through 20000 iterations at 60.67it/s.
- "physics": went through 20000 iterations at 38.50it/s.
- "graphics": went through 15861 iterations at 57.71it/s.

Of particular note, seven of the 21 keywords maxed out the number of iterations. They are: "transformers", "graph", "colorization", "attention", "explainable", and "physics". At the other end of the spectrum is "semi-supervised learning" which went through zero iterations either because it is not in the archive or it is but in a different format without the hypen.

In [6]:
```
raw_data = pd.DataFrame({
    'titles': all_titles,
    'abstracts': all_summaries,
    'terms': all_terms
})

raw_data.head()
```

Out[6]:

	titles	abstracts	terms
0	Reinforcement Learning from Multiple Sensors v...	In many scenarios, observations from more than...	[cs.LG]
1	Interventional Causal Representation Learning	Causal representation learning seeks to extrac...	[stat.ML, cs.LG]
2	Self-Supervised Node Representation Learning v...	Self-supervised node representation learning a...	[cs.LG]
3	Out-of-Distribution Representation Learning fo...	Time series classification is an important pro...	[cs.LG, cs.AI]
4	Trading Information between Latents in Hierarc...	Variational Autoencoders (VAEs) were originall...	[stat.ML, cs.CV, cs.IT, cs.LG, math.IT]

In [7]:
```
raw_data.shape #(64573, 3)
```

Out[7]:
```
(64573, 3)
```

In [8]:
```
# saving to csv file because pulling data takes a long while

    # well I didn't get it to the file folder but it did write.


# This is how you save a pandas dataframe and all the details associated with it.  You
raw_data.to_pickle('G:\\My Drive\\ADS-509_Final_Team_Project\\arxiv_data_2023_02_13.pk

# Nextime you want to use it.  All you have to do is read it as a pickle file. The fil
# raw_data = pd.read_pickle('G:\\My Drive\\ADS-509_Final_Team_Project\\arxiv_data_2023
```

Appendix B: Descriptive Statistics

Assignment 6.2: Preparing Data for Final Team Project

ADS 509: Final Project Code

Team 4

Zachariah Freitas and Brianne Bell

Instructions:

1. Set up a GitHub repository for your final project. Make it public, or add your instructor as a collaborator. Include in this repository the code to assemble your final project data set. The best practice is to place this data creation code in its own notebook.
2. In a separate notebook, calculate descriptive statistics on your final-project data. You are welcome to reuse code from earlier modules.

Deliverable:

- When you have completed this notebook, run all cells, print the notebook as a PDF, and submit the PDF in Blackboard.
- Commit and push your code, so your repo is up to date.
- Enter your GitHub link as "online text" in the Blackboard assignment.

In [1]:
```
# Import Libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from collections import Counter
from nltk.corpus import stopwords
from string import punctuation
from wordcloud import WordCloud
import textacy.preprocessing as tprep
from lexical_diversity import lex_div as ld

# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison

# Stopwords
sw = stopwords.words("english")
```

In [2]:
```
# Get Imported Data
raw_data = pd.read_pickle('G:\\My Drive\\ADS-509_Final_Team_Project\\arxiv_data_2023_6
raw_data
```

Out[2]:

	titles	abstracts	terms
0	Reinforcement Learning from Multiple Sensors v...	In many scenarios, observations from more than...	[cs.LG]
1	Interventional Causal Representation Learning	Causal representation learning seeks to extrac...	[stat.ML, cs.LG]
2	Self-Supervised Node Representation Learning v...	Self-supervised node representation learning a...	[cs.LG]
3	Out-of-Distribution Representation Learning fo...	Time series classification is an important pro...	[cs.LG, cs.AI]
4	Trading Information between Latents in Hierarc...	Variational Autoencoders (VAEs) were originall...	[stat.ML, cs.CV, cs.IT, cs.LG, math.IT]
...	...	...	...
64568	Plot 94 in ambiance X-Window	<PLOT > is a collection of routines to draw su...	[cs.CV, cs.GR]
64569	Automatic Face Recognition System Based on Loc...	We present an automatic face verification syst...	[cs.CV]
64570	Convexity Analysis of Snake Models Based on Ha...	This paper presents a convexity analysis for t...	[cs.CV, cs.GR, I.4; I.4.6;I.4.8]
64571	Semi-automatic vectorization of linear network...	A system for semi-automatic vectorization of l...	[cs.CV, cs.MM, I.4.6]
64572	Digital Color Imaging	This paper surveys current technology and rese...	[cs.CV, cs.GR, A.1;I.4,I.3.3,I.2.10;I.3.7;B.4.2]

64573 rows × 3 columns

In [3]:

```
# Helper Function - Descriptive Statistics
def descriptive_stats(tokens, top_n_tokens = 5, verbose=True) :
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical
        and num_tokens most common tokens. Return a list with the number of tokens, nu
        of unique tokens, lexical diversity, and number of characters.

    """
    if (len(tokens) == 0):
        raise ValueError("Can't work with empty text object.")

    # These are placeholder values.
    num_tokens = 1
    num_unique_tokens = 0
    avg_token_len = 0.0
    lexical_diversity = 0.0

    # Calculate statistics
    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    lexical_diversity = ld.ttr(tokens) # Simple TTR = len(Counter(text))/len(text)
    num_characters = sum([len(i) for i in tokens])
    avg_token_len = np.mean([len(w) for w in tokens])
```

```
    top_words = Counter(tokens).most_common(top_n_tokens)

    # Inline Printing
    if verbose:
        print(f"There are {num_tokens} tokens in the data.")
        print(f"There are {num_unique_tokens} unique tokens in the data.")
        print(f"There are {num_characters} characters in the data.")
        print(f"The average token length in the data is {avg_token_len:.3f}.")
        print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")

        # print the five most common tokens
        print(f"\n\nThe top {top_n_tokens} most common tokens")
        print(top_words)


    # Return Dictionary
    results = { 'tokens' : num_tokens,
            'unique_tokens' : num_unique_tokens,
            'avg_token_length' : avg_token_len,
            'lexical_diversity': lexical_diversity,
            'num_characters': num_characters,
            'top_words': top_words}

    return(results)
```

In [4]:

```
# Helper Functions - Cleaning data

def normalize(text):
    text = tprep.normalize.hyphenated_words(text)
    text = tprep.normalize.quotation_marks(text)
    text = tprep.normalize.unicode(text)
    text = tprep.remove.accents(text)
    return text

def remove_punctuation(text, punct_set=punctuation) :
    """This function removes punctuation from a string."""
    return("".join([ch for ch in text if ch not in punct_set]))

def tokenize(text) :
    """ Splitting on whitespace rather than the book's tokenize function. That
        function will drop tokens like '#hashtag' or '2A', which we need for Twitter.

    # modify this function to return tokens
    return text.lower().strip().split()

def remove_stop(tokens) :
    """This function removes stopwords from a list of tokens."""
    return([t for t in tokens if t.lower() not in sw])

def prepare(text, pipeline) :
    """ This fuction manages and executes other functions like a pipline. """
    tokens = str(text)

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

In [5]:
```
# Helper Function - Flatten a List of Lists
def flatten_lists(list_of_lists):
    """This function flattens a list of lists into a single list."""
    return [i for s in list_of_lists for i in s]
```

In [6]:
```
# Clean and Tokenize Data
df = raw_data.copy()

df["titles_abs"] = df["titles"] + df["abstracts"]

my_pipeline = [normalize, remove_punctuation, tokenize, remove_stop]
df["titles_tokens"] = df["titles"].apply(prepare, pipeline=my_pipeline)
df["abstract_tokens"] = df["abstracts"].apply(prepare, pipeline=my_pipeline)
df["titles_abs_tokens"] = df["titles_abs"].apply(prepare, pipeline=my_pipeline)

df
```

Out[6]:

	titles	abstracts	terms	titles_abs	titles_tokens	abstra
0	Reinforcement Learning from Multiple Sensors v...	In many scenarios, observations from more than...	[cs.LG]	Reinforcement Learning from Multiple Sensors v...	[reinforcement, learning, multiple, sensors, v...	ob one, s
1	Interventional Causal Representation Learning	Causal representation learning seeks to extrac...	[stat.ML, cs.LG]	Interventional Causal Representation LearningC...	[interventional, causal, representation, learn...	repr learr
2	Self-Supervised Node Representation Learning v...	Self-supervised node representation learning a...	[cs.LG]	Self-Supervised Node Representation Learning v...	[selfsupervised, node, representation, learnin...	[selfs repr
3	Out-of-Distribution Representation Learning fo...	Time series classification is an important pro...	[cs.LG, cs.AI]	Out-of-Distribution Representation Learning fo...	[outofdistribution, representation, learning, ...	[ti cla
4	Trading Information between Latents in Hierarc...	Variational Autoencoders (VAEs) were originall...	[stat.ML, cs.CV, cs.IT, cs.LG, math.IT]	Trading Information between Latents in Hierarc...	[trading, information, latents, hierarchical, ...	[v aut vaes,
...	...	...	...	...	...	
64568	Plot 94 in ambiance X-Window	<PLOT > is a collection of routines to draw su...	[cs.CV, cs.GR]	Plot 94 in ambiance X-Window<PLOT > is a colle...	[plot, 94, ambiance, xwindow]	[plot, rout s
64569	Automatic Face Recognition System Based on Loc...	We present an automatic face verification syst...	[cs.CV]	Automatic Face Recognition System Based on Loc...	[automatic, face, recognition, system, based, ...	auto v
64570	Convexity Analysis of Snake Models Based on Ha...	This paper presents a convexity analysis for t...	[cs.CV, cs.GR, I.4; I.4.6;I.4.8]	Convexity Analysis of Snake Models Based on Ha...	[convexity, analysis, snake, models, based, ha...	[paper
64571	Semi-automatic vectorization of linear network...	A system for semi-automatic vectorization of l...	[cs.CV, cs.MM, I.4.6]	Semi-automatic vectorization of linear network...	[semiautomatic, vectorization, linear, network...	semi vec
64572	Digital Color Imaging	This paper surveys current technology and rese...	[cs.CV, cs.GR, A.1;I.4,I.3.3,I.2.10;I.3.7;B.4.2]	Digital Color ImagingThis paper surveys curren...	[digital, color, imaging]	[pap t

64573 rows × 7 columns

In [7]:
```
# Statistics on Titles
title_results = descriptive_stats(flatten_lists(df['titles_tokens']))
```

```
There are 470005 tokens in the data.
There are 31401 unique tokens in the data.
There are 3829797 characters in the data.
The average token length in the data is 8.148.
The lexical diversity is 0.067 in the data.


The top 5 most common tokens
[('learning', 19062), ('detection', 6548), ('deep', 6117), ('neural', 5997), ('networ
ks', 5569)]
```

In [8]:
```
# Statistics on Abstracts
abstract_results = descriptive_stats(flatten_lists(df['abstract_tokens']))
```

```
There are 7296290 tokens in the data.
There are 123974 unique tokens in the data.
There are 55306651 characters in the data.
The average token length in the data is 7.580.
The lexical diversity is 0.017 in the data.


The top 5 most common tokens
[('learning', 79692), ('data', 57018), ('model', 52447), ('methods', 41571), ('model
s', 39819)]
```

In [9]:
```
# Statistics on Titles & Abstractsabstract_results =
title_abstract_results = descriptive_stats(flatten_lists(df['titles_abs_tokens']))
```

```
There are 7726299 tokens in the data.
There are 155200 unique tokens in the data.
There are 59201115 characters in the data.
The average token length in the data is 7.662.
The lexical diversity is 0.020 in the data.


The top 5 most common tokens
[('learning', 92779), ('data', 58746), ('model', 54062), ('image', 44154), ('method
s', 42002)]
```

Word Cloud Plotting

In [10]:
```
# Helper Function - Plot Word Cloud

def wordcloud(word_freq, title=None, max_words=200, stopwords=None):
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical
        and num_tokens most common tokens. Return a list with the number of tokens, nu
        of unique tokens, lexical diversity, and number of characters.

    """
    wc = WordCloud(width=800, height=400,
                   background_color= "black", colormap="Paired",
                   max_font_size=150, max_words=max_words)
```

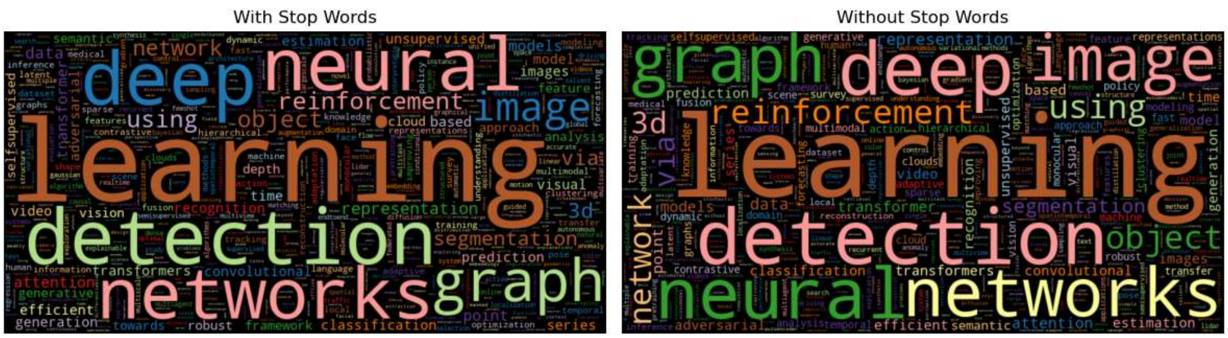
```
    # convert data frame into dict
    if type(word_freq) == pd.Series:
        counter = Counter(word_freq.fillna(0).to_dict())
    else:
        counter = word_freq

    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                              if token not in stopwords}
    wc.generate_from_frequencies(counter)

    plt.title(title)

    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")


def count_words(df, column='tokens', preprocess=None, min_freq=2):
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical
        and num_tokens most common tokens. Return a list with the number of tokens, nu
        of unique tokens, lexical diversity, and number of characters.
    """
    # process tokens and update counter
    def update(doc):
        tokens = doc if preprocess is None else preprocess(doc)
        counter.update(tokens)

    # create counter and run through all data
    counter = Counter()
    df[column].map(update)

    # transform counter into data frame
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    freq_df = freq_df.query('freq >= @min_freq')
    freq_df.index.name = 'token'

    return freq_df.sort_values('freq', ascending=False)



def plot_wc(wordcloud_df):
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical
        and num_tokens most common tokens. Return a list with the number of tokens, nu
        of unique tokens, lexical diversity, and number of characters.
    """
    plt.figure(figsize=(12,4))
    plt.subplot(1,2,1)###
    wordcloud(wordcloud_df['freq'], max_words=1000)
    plt.title("With Stop Words")

    plt.subplot(1,2,2)###
    wordcloud(wordcloud_df['freq'], max_words=1000, stopwords=sw)
    plt.title("Without Stop Words")
    plt.tight_layout()###
```

Corpus Word Cloud

Title Word Cloud

With stopwords and without stopwords.

In [11]:
```
wordcloud_df = count_words(df, column='titles_tokens')
plot_wc(wordcloud_df)
```



Title word cloud differences between with stop words and without shows that removing stop words increases the number of recurring words. This can be seen above where there are more large font words in the 'Without Stop Words' wordcloud (~10) than there are in the wordcloud with stopwords (~6).

Abstract Word Cloud

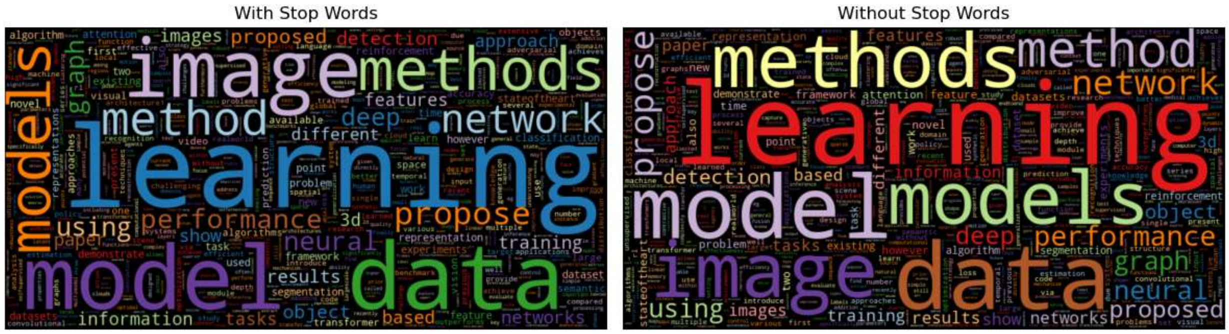
With stopwords and without stopwords.

In [12]:
```
wordcloud_df = count_words(df, column='abstract_tokens')
plot_wc(wordcloud_df)
```

Title & Abstract Word Cloud

With stopwords and without stopwords.

In [13]:
```
wordcloud_df = count_words(df, column='titles_abs_tokens')
plot_wc(wordcloud_df)
```



Looking at the combination of abstract and title tokens, both the wordclouds (with and without stopwords) appear very similar. The most common tokens are 'learning', 'model', 'data', 'image', 'method', 'models', and 'network'. These are heavily leaning towards data science topics so it is likely that our models will be heavily populated by these and could impact modeling performance.

Appendix C:Topic Modeling to Classification Modeling

ADS 509: Final Project Code

Team 4

Zachariah Freitas and Brianne Bell

Since our data is organized into groups with overlap of groups ('terms') where each entry has 1-11 groups, we will first determine our own topic groups through topic modeling.

Once our topic modeling is complete, we will use those results to develop classification models. These will be the multinomial Naive Bayes classifier and multinomial Logistic Regression classifier models. We will then report accuracy of our models against the hold out test set.

In [2]:
```
# Import Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
from matplotlib import pyplot as plt
from collections import Counter
from nltk.corpus import stopwords
from string import punctuation
from wordcloud import WordCloud
import textacy.preprocessing as tprep
from lexical_diversity import lex_div as ld

from ast import literal_eval
from itertools import islice, cycle

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, TruncatedSVD, LatentDirichletAllocation
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix

import warnings
warnings.filterwarnings('ignore')
```

Helper Functions

In [3]:
```
# Add additional helper functions

# Stopwords
sw = stopwords.words("english")

# Some punctuation variations
punctuation = set(punctuation) # speeds up comparison
# tw_punct = punctuation - {"#"} # Do this if there is punctuation you would like to i

# Helper Functions - Cleaning data

def normalize(text):
    text = tprep.normalize.hyphenated_words(text)
    text = tprep.normalize.quotation_marks(text)
    text = tprep.normalize.unicode(text)
    text = tprep.remove.accents(text)
    return text

def remove_punctuation(text, punct_set=punctuation) :
    """This function removes punctuation from a string."""
    return("".join([ch for ch in text if ch not in punct_set]))

def tokenize(text) :
    """ Splitting on whitespace rather than the book's tokenize function. That
        function will drop tokens like '#hashtag' or '2A', which we need for Twitter.

    # modify this function to return tokens
    return text.lower().strip().split()

def remove_stop(tokens) :
    """This function removes stopwords from a list of tokens."""
    return([t for t in tokens if t.lower() not in sw])

def detokenize(list) :
    """ Returning back a string from a list of tokens"""

    # modify this function to return string from tokens
    return(" ".join([str(e) for e in list]))

def prepare(text, pipeline) :
    """ This fuction manages and executes other functions like a pipline. """
    tokens = str(text)

    for transform in pipeline :
        tokens = transform(tokens)

    return(tokens)
```

Data ETL and EDA

In [4]:
```
# Get Imported Data
arxiv_data = pd.read_pickle('G:\\My Drive\\ADS-509_Final_Team_Project\\arxiv_data_202
arxiv_data
```

Out[4]:

	titles	abstracts	terms
0	Reinforcement Learning from Multiple Sensors v...	In many scenarios, observations from more than...	[cs.LG]
1	Interventional Causal Representation Learning	Causal representation learning seeks to extrac...	[stat.ML, cs.LG]
2	Self-Supervised Node Representation Learning v...	Self-supervised node representation learning a...	[cs.LG]
3	Out-of-Distribution Representation Learning fo...	Time series classification is an important pro...	[cs.LG, cs.AI]
4	Trading Information between Latents in Hierarc...	Variational Autoencoders (VAEs) were originall...	[stat.ML, cs.CV, cs.IT, cs.LG, math.IT]
...	...	...	...
64568	Plot 94 in ambiance X-Window	<PLOT > is a collection of routines to draw su...	[cs.CV, cs.GR]
64569	Automatic Face Recognition System Based on Loc...	We present an automatic face verification syst...	[cs.CV]
64570	Convexity Analysis of Snake Models Based on Ha...	This paper presents a convexity analysis for t...	[cs.CV, cs.GR, I.4; I.4.6;I.4.8]
64571	Semi-automatic vectorization of linear network...	A system for semi-automatic vectorization of l...	[cs.CV, cs.MM, I.4.6]
64572	Digital Color Imaging	This paper surveys current technology and rese...	[cs.CV, cs.GR, A.1;I.4,I.3.3,I.2.10;I.3.7;B.4.2]

64573 rows × 3 columns

In [5]:

```
# Data Review
print(f"There are {len(arxiv_data)} rows in the dataset.\n")

total_duplicate_titles = sum(arxiv_data["titles"].duplicated())
print(f"There are {total_duplicate_titles} duplicated titles.\n")

dedup_arxiv_data = arxiv_data[~arxiv_data["titles"].duplicated()].reset_index(drop=Tru
print(f"There are {len(dedup_arxiv_data)} rows in the deduplicated dataset.\n")

# There are some terms with occurrence as low as 1.
occur = sum(dedup_arxiv_data["terms"].value_counts() == 1)
print(f"There are {occur} terms with an occurrence of 1.\n")

# How many unique terms?
unique_terms = dedup_arxiv_data["terms"].astype('str').nunique()
print(f"There are {unique_terms} unique terms.\n")
```

```
There are 64573 rows in the dataset.

There are 14237 duplicated titles.

There are 50336 rows in the deduplicated dataset.

There are 2907 terms with an occurrence of 1.

There are 3961 unique terms.
```

Normalize and Clean Text

In [6]:

```
# Clean and Tokenize Data
df = dedup_arxiv_data.copy()

df["summaries"] = df["titles"] + df["abstracts"]

my_pipeline = [normalize, remove_punctuation, tokenize, remove_stop, detokenize]
df["summaries"] = df["summaries"].apply(prepare, pipeline=my_pipeline)

df
```

Out[6]:

	titles	abstracts	terms	summaries
0	Reinforcement Learning from Multiple Sensors v...	In many scenarios, observations from more than...	[cs.LG]	reinforcement learning multiple sensors via jo...
1	Interventional Causal Representation Learning	Causal representation learning seeks to extrac...	[stat.ML, cs.LG]	interventional causal representation learningc...
2	Self-Supervised Node Representation Learning v...	Self-supervised node representation learning a...	[cs.LG]	selfsupervised node representation learning vi...
3	Out-of-Distribution Representation Learning fo...	Time series classification is an important pro...	[cs.LG, cs.AI]	outofdistribution representation learning time...
4	Trading Information between Latents in Hierarc...	Variational Autoencoders (VAEs) were originall...	[stat.ML, cs.CV, cs.IT, cs.LG, math.IT]	trading information latents hierarchical varia...
...	...	...	...	...
50331	Plot 94 in ambiance X-Window	<PLOT > is a collection of routines to draw su...	[cs.CV, cs.GR]	plot 94 ambiance xwindowplot collection routin...
50332	Automatic Face Recognition System Based on Loc...	We present an automatic face verification syst...	[cs.CV]	automatic face recognition system based local ...
50333	Convexity Analysis of Snake Models Based on Ha...	This paper presents a convexity analysis for t...	[cs.CV, cs.GR, I.4; I.4.6;I.4.8]	convexity analysis snake models based hamilton...
50334	Semi-automatic vectorization of linear network...	A system for semi-automatic vectorization of l...	[cs.CV, cs.MM, I.4.6]	semiautomatic vectorization linear networks ra...
50335	Digital Color Imaging	This paper surveys current technology and rese...	[cs.CV, cs.GR, A.1;I.4,I.3.3,I.2.10;I.3.7;B.4.2]	digital color imagingthis paper surveys curren...

50336 rows × 4 columns

Topic Modeling

In [7]:

```
# Helper Functions
# This function comes from the BTAP repo.

def display_topics(model, features, no_top_words=5):
    for topic, words in enumerate(model.components_):
        total = words.sum()
        largest = words.argsort()[::-1] # invert sort order
        print("\nTopic %02d" % topic)
        for i in range(0, no_top_words):
            print("  %s (%2.2f)" % (features[largest[i]], abs(words[largest[i]]*100.0/
```

In [8]:
```
# Count the words
count_text_vectorizer = CountVectorizer(stop_words=list(sw), min_df=0, max_df=0.3)
count_text_vectors = count_text_vectorizer.fit_transform(df['summaries'])
print(count_text_vectors.shape)
```

```
(50336, 155123)
```

In [9]:
```
# Get TF-IDF
tfidf_text_vectorizer = TfidfVectorizer(stop_words=list(sw), min_df=0, max_df=0.3)
tfidf_text_vectors = tfidf_text_vectorizer.fit_transform(df['summaries'])
print(tfidf_text_vectors.shape)
```

```
(50336, 155123)
```

Fitting a Non-Negative Matrix Factorization Model - NMF

In [10]:
```
nmf_text_model = NMF(n_components=5, random_state=314)
W_nmf_text_matrix = nmf_text_model.fit_transform(tfidf_text_vectors)
H_nmf_text_matrix = nmf_text_model.components_
```

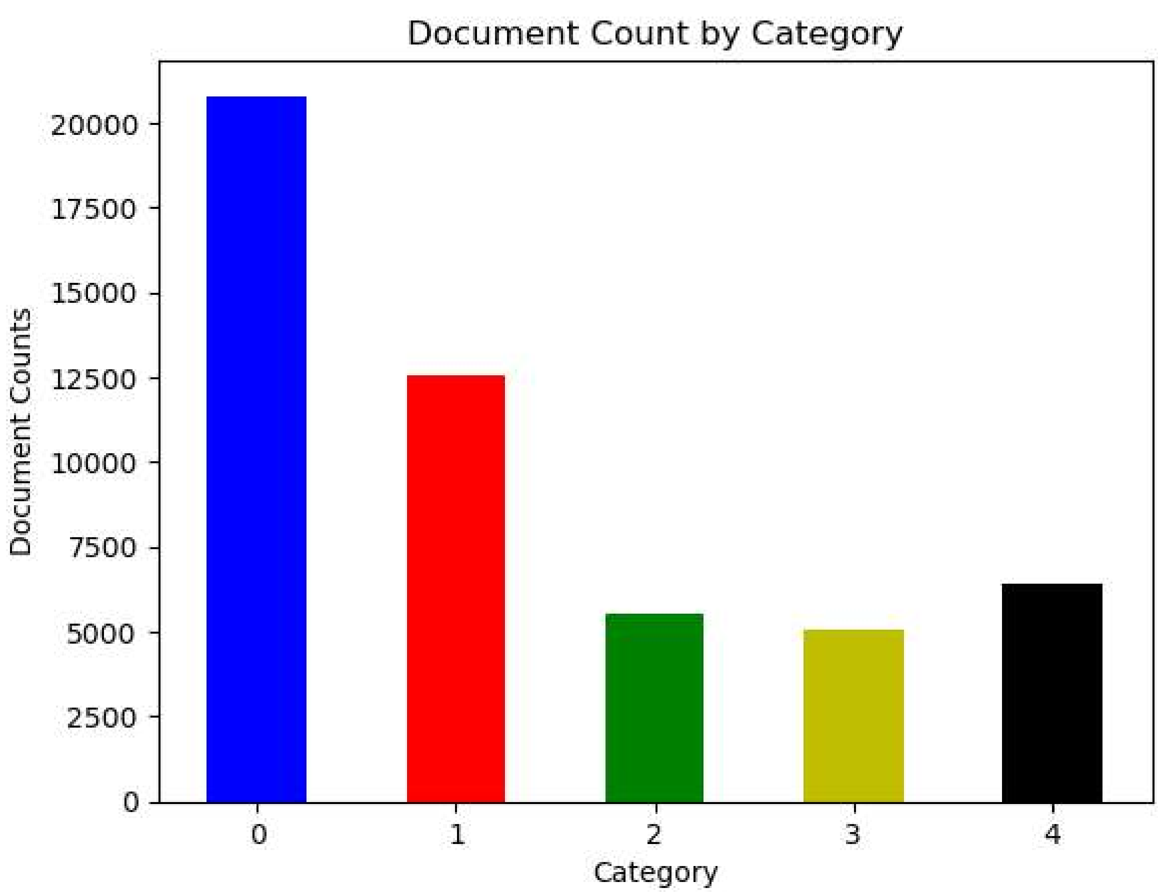
In [11]:
```
display_topics(nmf_text_model, tfidf_text_vectorizer.get_feature_names_out(), no_top_w
```

```
Topic 00
  image (0.75)
  images (0.45)
  segmentation (0.42)
  training (0.30)
  deep (0.27)
  attention (0.26)
  visual (0.26)
  features (0.26)
  semantic (0.26)
  neural (0.25)

Topic 01
  reinforcement (0.91)
  policy (0.84)
  rl (0.76)
  algorithm (0.45)
  algorithms (0.44)
  agents (0.38)
  reward (0.38)
  agent (0.36)
  function (0.33)
  control (0.33)

Topic 02
  graph (3.21)
  graphs (0.96)
  node (0.87)
  neural (0.73)
  gnns (0.73)
  networks (0.65)
  nodes (0.56)
  gnn (0.45)
  representation (0.40)
  structure (0.38)

Topic 03
  3d (3.23)
  point (2.73)
  cloud (1.58)
  depth (1.15)
  clouds (1.10)
  shape (0.75)
  2d (0.68)
  estimation (0.63)
  pose (0.58)
  points (0.52)

Topic 04
  object (2.83)
  detection (2.64)
  objects (0.86)
  detectors (0.49)
  tracking (0.46)
  detector (0.45)
  dataset (0.38)
  anomaly (0.37)
  bounding (0.35)
  video (0.34)
```

In [13]:

```
# Evaluate the Categories
df_new = df.copy()

# Get the top predicted topic
predicted_topics = [np.argsort(each)[::-1][0] for each in W_nmf_text_matrix]

# Add to the df
df_new['nmf_pred_topic_num'] = predicted_topics

# Create Chart
my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(df)))
bar_plot = df_new.groupby('nmf_pred_topic_num').size()
ax = bar_plot.plot(kind="bar", stacked=True, rot=0, color=my_colors, title='Document C
ax.set_xlabel("Category")
ax.set_ylabel("Document Counts")
display(ax)
print('\n')
```

```
<AxesSubplot:title={'center':'Document Count by Category'}, xlabel='Category', ylabel
='Document Counts'>
```



Fitting an LSA Model - LSA

In [14]:

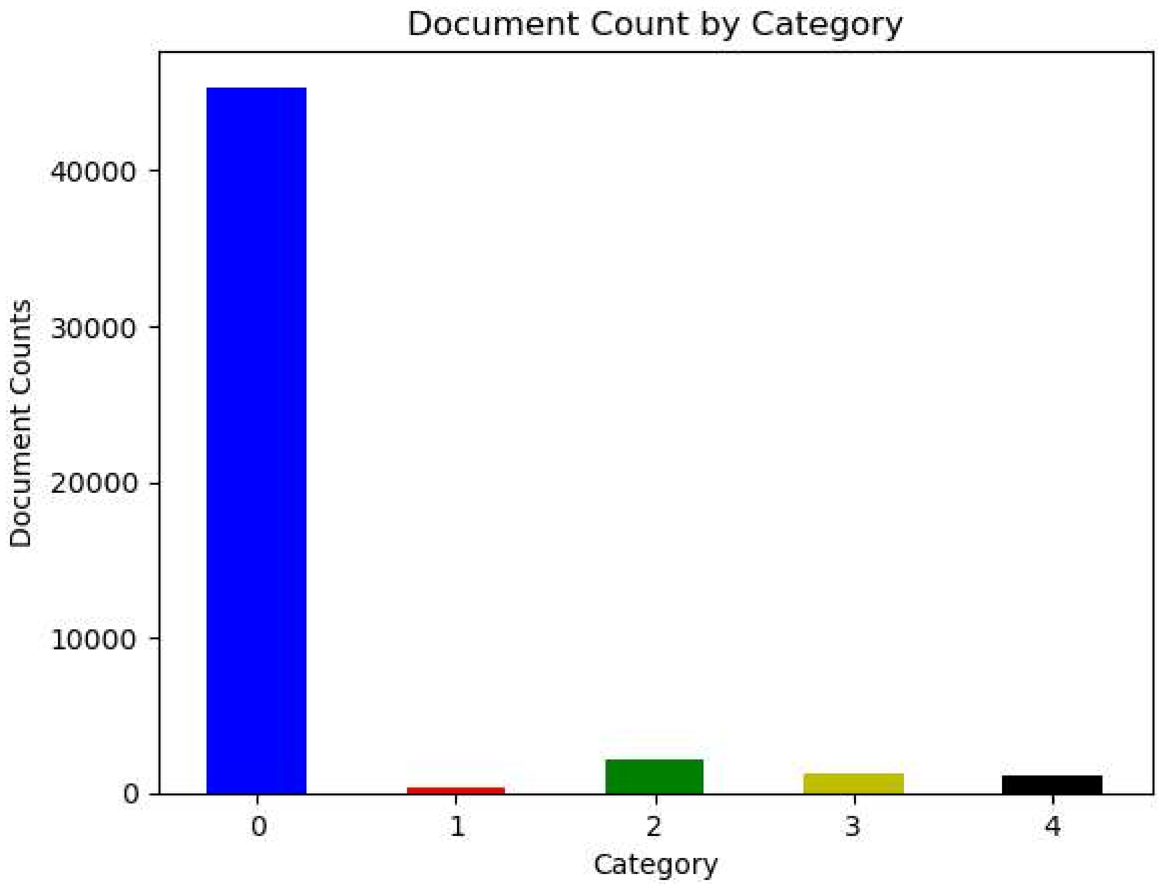
```
# Your code here
lsa_text_model = TruncatedSVD(n_components=5, n_iter=14, random_state=42)
W_lsa_text_matrix = lsa_text_model.fit_transform(tfidf_text_vectors)
H_lsa_text_matrix = lsa_text_model.components_
```

In [15]:
```
display_topics(lsa_text_model, tfidf_text_vectorizer.get_feature_names_out(), no_top_w
```

```
Topic 00
  image (0.34)
  object (0.29)
  detection (0.27)
  graph (0.27)
  3d (0.26)

Topic 01
  3d (3.69)
  object (3.16)
  detection (2.36)
  point (2.20)
  segmentation (1.69)

Topic 02
  graph (13.99)
  graphs (4.07)
  node (3.86)
  gnns (3.19)
  neural (2.77)

Topic 03
  point (25.82)
  3d (25.61)
  cloud (15.06)
  graph (11.65)
  clouds (10.29)

Topic 04
  detection (52.93)
  object (52.39)
  graph (15.39)
  objects (12.99)
  detectors (9.87)
```

In [16]:
```
# Evaluate the Categories

# Get the top predicted topic
predicted_topics = [np.argsort(each)[::-1][0] for each in W_lsa_text_matrix]

# Add to the df
df_new['lsa_pred_topic_num'] = predicted_topics

# Create Chart
my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(df)))
bar_plot = df_new.groupby('lsa_pred_topic_num').size()
ax = bar_plot.plot(kind="bar", stacked=True, rot=0, color=my_colors, title='Document C
ax.set_xlabel("Category")
ax.set_ylabel("Document Counts")
display(ax)
print('\n')
```

```
<AxesSubplot:title={'center':'Document Count by Category'}, xlabel='Category', ylabel
='Document Counts'>
```



Fitting an LDA Model - LDA

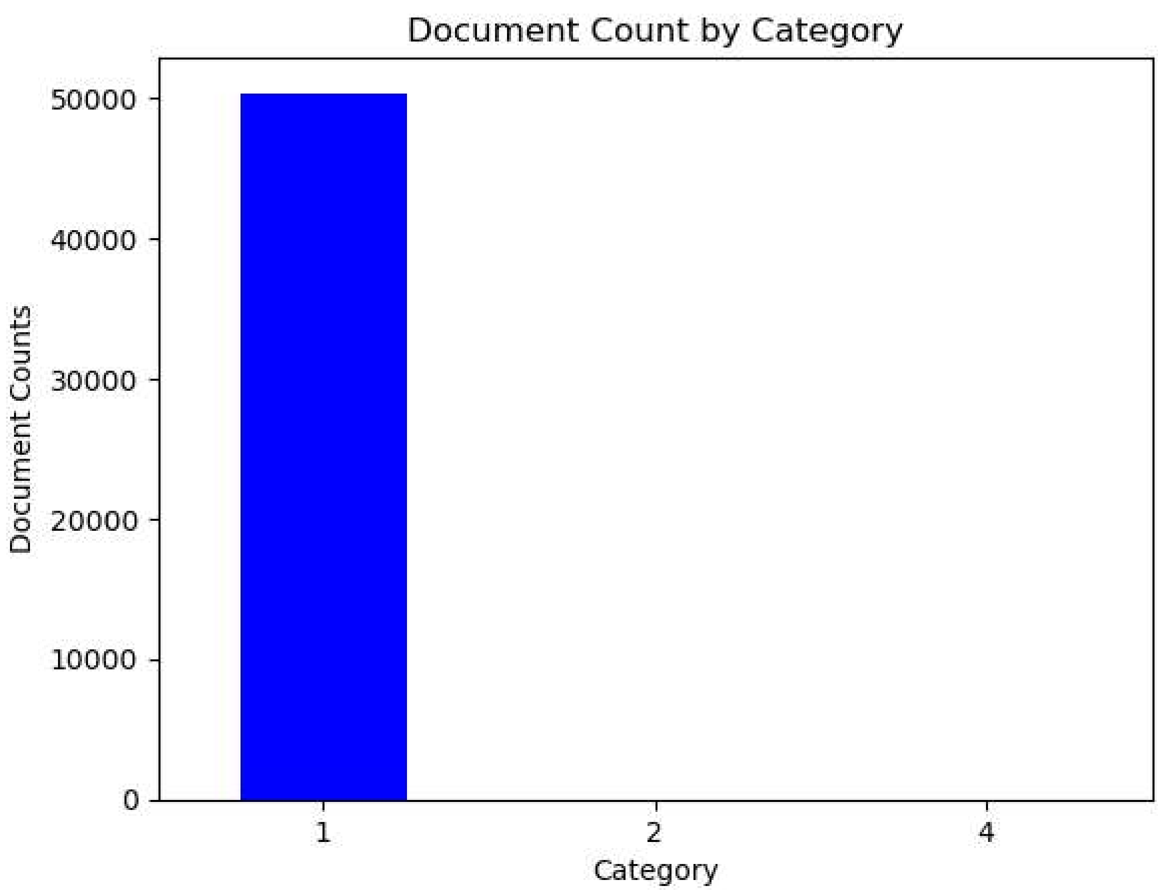
In [17]:
```
# Fit your LDA model here

lda_text_model = LatentDirichletAllocation(n_components=5, random_state=0)
W_lda_text_matrix = lda_text_model.fit_transform(tfidf_text_vectors)
H_lda_text_matrix = lda_text_model.components_
```

In [18]:
```
# Call `display_topics` on your fitted model here
display_topics(lda_text_model, tfidf_text_vectorizer.get_feature_names_out(), no_top_w
```

```
Topic 00
  font (0.02)
  glyph (0.01)
  glyphs (0.01)
  nst (0.01)
  bolt (0.01)
  ncdes (0.01)
  vitae (0.01)
  winograd (0.01)
  fingerspelling (0.01)
  spill (0.01)

Topic 01
  image (0.27)
  graph (0.22)
  object (0.22)
  detection (0.22)
  neural (0.22)
  deep (0.21)
  training (0.19)
  images (0.19)
  3d (0.19)
  networks (0.19)

Topic 02
  graphon (0.02)
  frl (0.01)
  de (0.01)
  cmi (0.01)
  whittle (0.01)
  un (0.01)
  graphcl (0.01)
  fmp (0.01)
  chromosomes (0.01)
  fin (0.01)

Topic 03
  spns (0.02)
  vfl (0.01)
  microexpression (0.01)
  mitotic (0.01)
  dpps (0.01)
  htr (0.01)
  capsnets (0.01)
  bangla (0.01)
  microexpressions (0.01)
  mcs (0.01)

Topic 04
  wsi (0.02)
  videoqa (0.02)
  wsis (0.02)
  msrvtt (0.02)
  videolanguage (0.02)
  videotext (0.02)
  garment (0.02)
  msvd (0.01)
  rainfall (0.01)
  textvqa (0.01)
```

In [19]:
```
# Evaluate the Categories

# Get the top predicted topic
predicted_topics = [np.argsort(each)[::-1][0] for each in W_lda_text_matrix]

# Add to the df
df_new['lda_pred_topic_num'] = predicted_topics

# Create Chart
my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(df)))
bar_plot = df_new.groupby('lda_pred_topic_num').size()
ax = bar_plot.plot(kind="bar", stacked=True, rot=0, color=my_colors, title='Document (
ax.set_xlabel("Category")
ax.set_ylabel("Document Counts")
display(ax)
print('\n')
```

```
<AxesSubplot:title={'center':'Document Count by Category'}, xlabel='Category', ylabel
='Document Counts'>
```



Topic Model Summary

Reviewing the output of each of our Topic Models we found that the NMF model results made the most sense. Both In the makeup of their content and in the distribution of documents. Moving forward, we recommend the use of the NMF categories for our predictive classification modeling.

Word Cloud

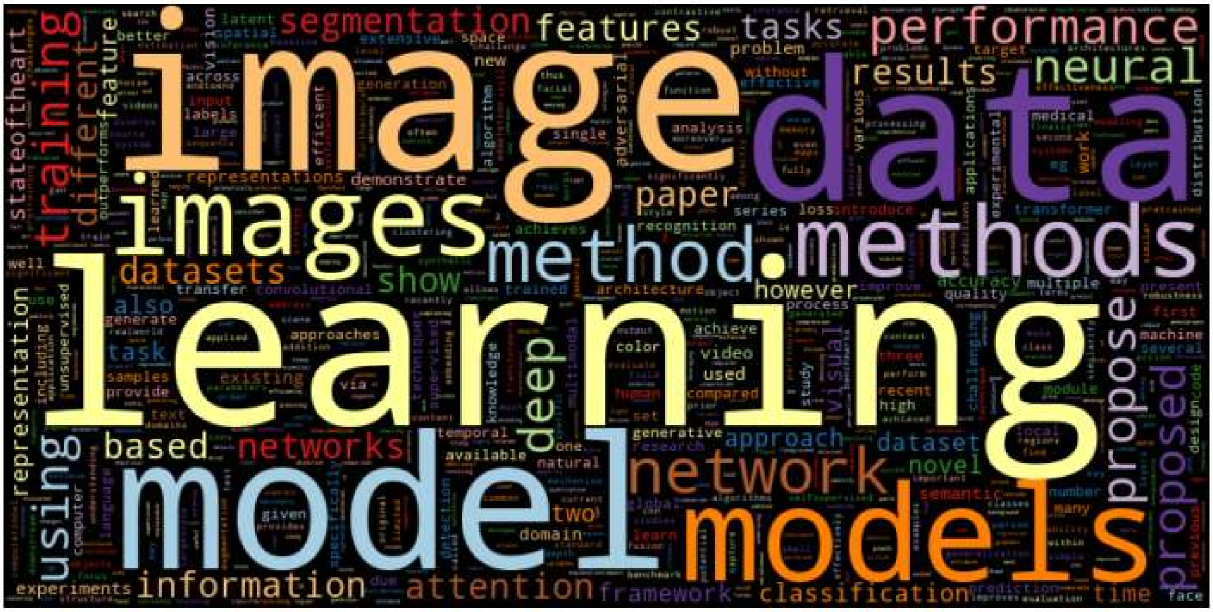
Here we wanted to show a word cloud of each of the categories from our preferred Topic Model, NMF.

In [20]:
```
# plot helper function

def wordcloud(word_freq, title=None, max_words=200, stopwords=None):

    wc = WordCloud(width=800, height=400,
                   background_color= "black", colormap="Paired",
                   max_font_size=150, max_words=max_words)

    # convert data frame into dict
    if type(word_freq) == pd.Series:
        counter = Counter(word_freq.fillna(0).to_dict())
    else:
        counter = word_freq

    # filter stop words in frequency counter
    if stopwords is not None:
        counter = {token:freq for (token, freq) in counter.items()
                              if token not in stopwords}
    wc.generate_from_frequencies(counter)

    plt.title(title)

    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")


def count_words(df, column='summaries', preprocess=None, min_freq=2):

    # process tokens and update counter
    def update(doc):
        tokens = doc if tokenize is None else tokenize(doc)
        counter.update(tokens)

    # create counter and run through all data
    counter = Counter()
    df[column].map(update)

    # transform counter into data frame
    freq_df = pd.DataFrame.from_dict(counter, orient='index', columns=['freq'])
    freq_df = freq_df.query('freq >= @min_freq')
    freq_df.index.name = 'token'

    return freq_df.sort_values('freq', ascending=False)


def plot_wc(wordcloud_df, title=None):
    plt.figure(figsize=(12,4))
    wordcloud(wordcloud_df['freq'], title=title, max_words=1000, stopwords=sw)
    plt.tight_layout()###
```

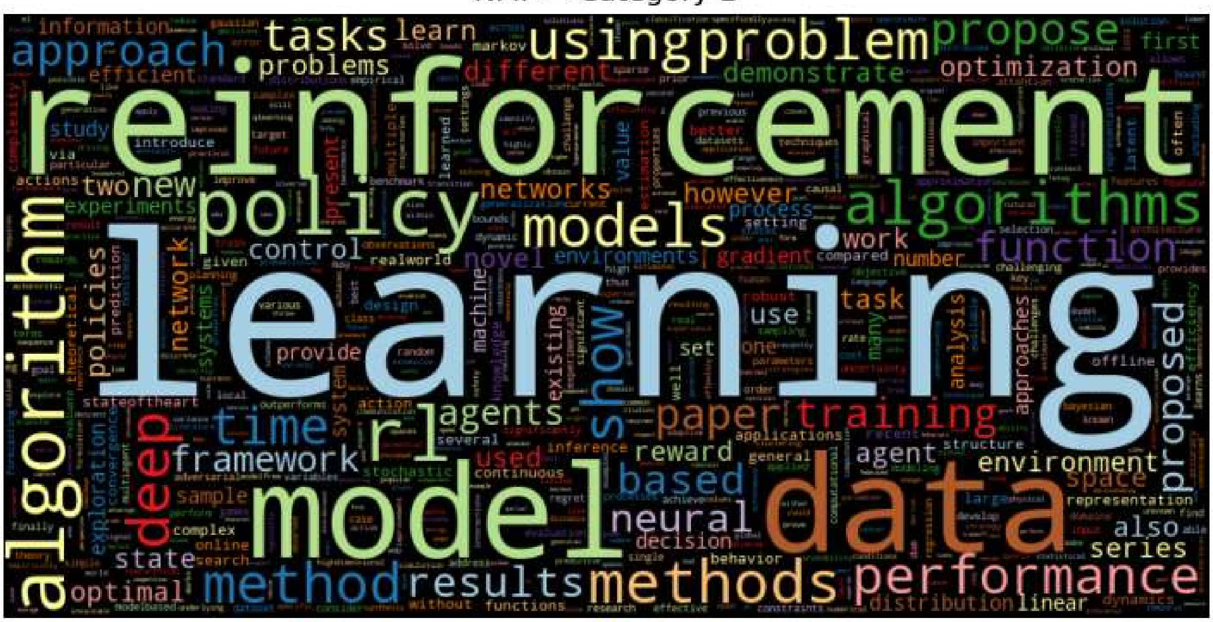
In [21]:

```
# Plot A Word Cloud for Each NMF Topic Category
cat_nums = [0,1,2,3,4]
for c in cat_nums:
    wordcloud_df = count_words(df_new.loc[df_new['nmf_pred_topic_num'] ==c])
    plot_wc(wordcloud_df, title = f'NMF = Category {c}')
```

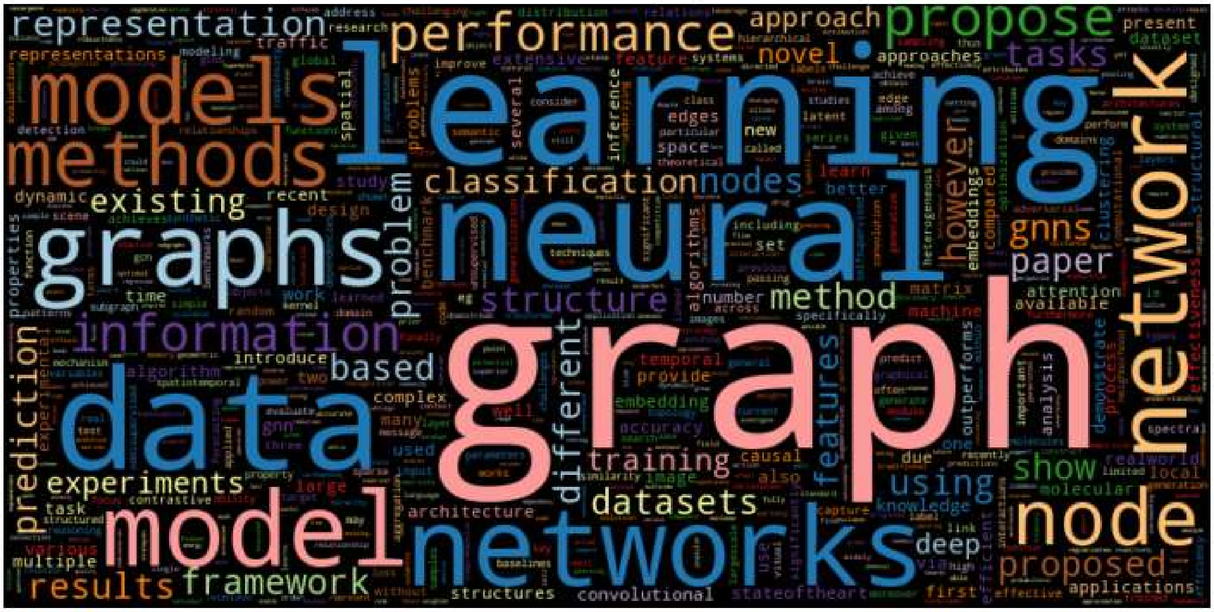
NMF = Category 0



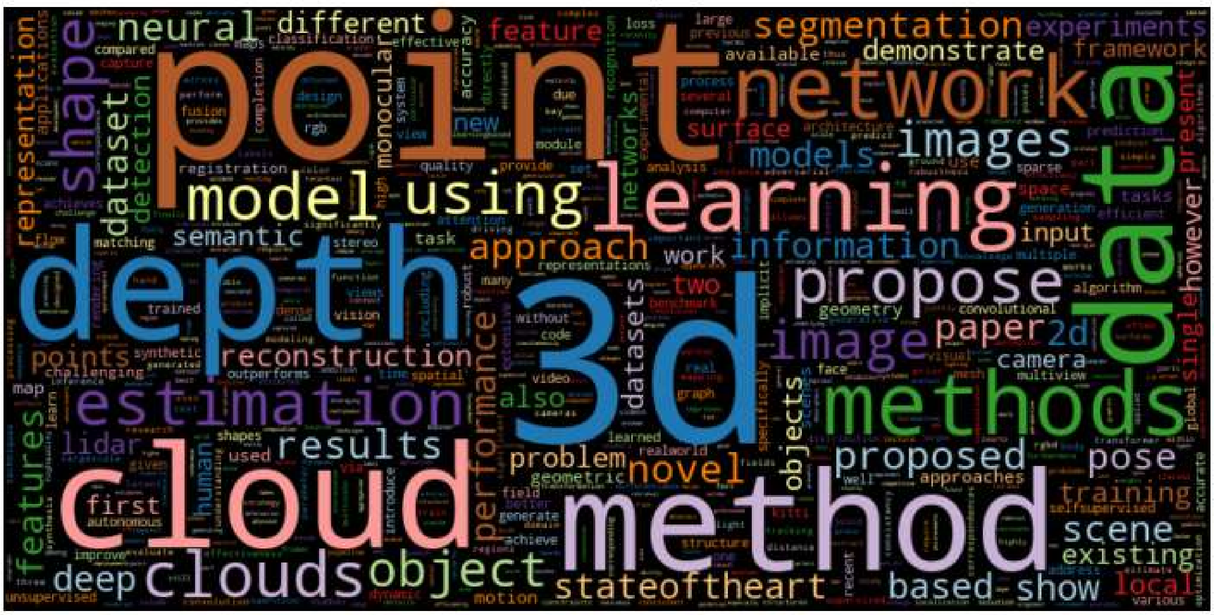
NMF = Category 1



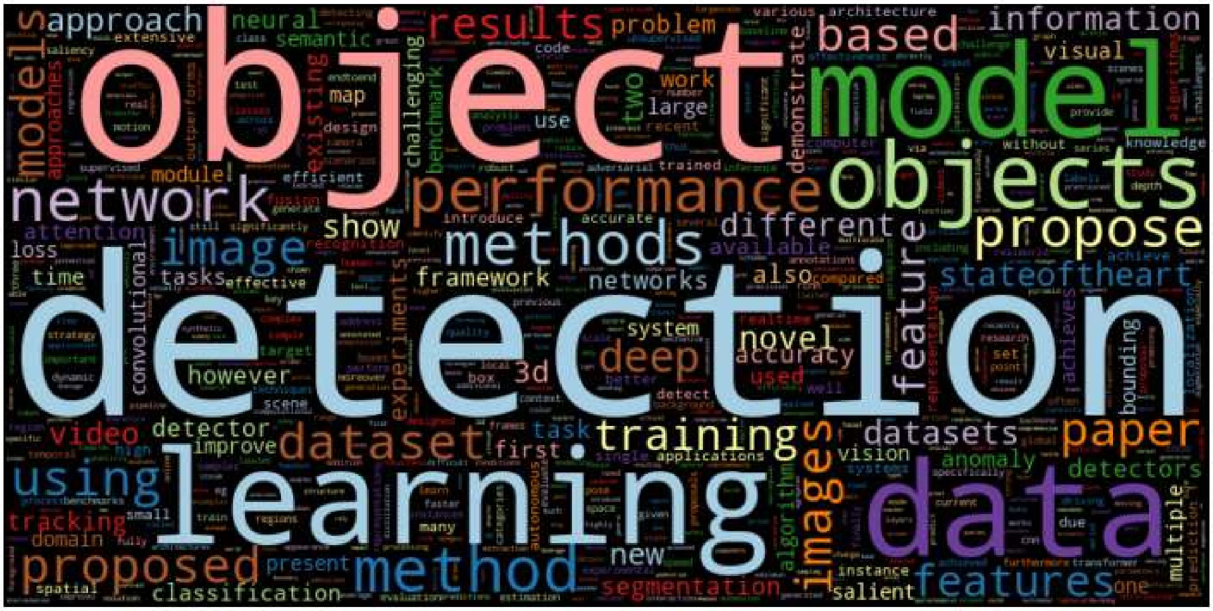
NMF = Category 2



NMF = Category 3



NMF = Category 4



Test Train Split

In [22]:
```python
y = pd.get_dummies(df_new['nmf_pred_topic_num'], prefix='cat_', dtype=float)
cat_col_names = list(y.columns)

print(cat_col_names)
y
```

```
['cat__0', 'cat__1', 'cat__2', 'cat__3', 'cat__4']
```

Out[22]:

	cat__0	cat__1	cat__2	cat__3	cat__4
0	0.0	1.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...
50331	0.0	1.0	0.0	0.0	0.0
50332	1.0	0.0	0.0	0.0	0.0
50333	0.0	0.0	0.0	1.0	0.0
50334	1.0	0.0	0.0	0.0	0.0
50335	1.0	0.0	0.0	0.0	0.0

50336 rows × 5 columns

In [23]:
```python
X_train, X_test, y_train, y_test = train_test_split(df['summaries'],
                                                    pd.get_dummies(df_new['nmf_pred_to
                                                    test_size=0.3,
```

```
                                                    random_state=43,
                                                    shuffle=True)
vectorizer = TfidfVectorizer(strip_accents='unicode', analyzer='word', ngram_range=(1,
vectorizer.fit(X_train)

X_train = vectorizer.transform(X_train)
X_test = vectorizer.transform(X_test)
```

Classification Modeling

Here we try two approaches for classification modeling. The first is Logistic Regression, and the second is Multinomial Naïve Bayes.

Logistic Regression

In [24]:

```
# Logistic Regression

LR_pipeline = Pipeline([('clf', OneVsRestClassifier(LogisticRegression(solver='sag'),
accuracy = 0
for category in cat_col_names:
    print('\n**Processing {} titles...**'.format(category))

    # Training logistic regression model on train data
    LR_pipeline.fit(X_train, y_train[category])

    # calculating test accuracy
    prediction = LR_pipeline.predict(X_test)
    accuracy = accuracy + accuracy_score(y_test[category], prediction)
    print('AUC ROC score is {}'.format(roc_auc_score(y_test[category],prediction)))
    conf_mat = confusion_matrix(
        y_test[category], prediction
        )
    print(conf_mat)
    print("\n")
    print("-"*40)

print('Test averaged AUC ROC is {}'.format(accuracy/len(cat_col_names)))
```

```
**Processing cat__0 titles...**
AUC ROC score is 0.9432332033456013
[[8285  580]
 [ 300 5936]]


----------------------------------------

**Processing cat__1 titles...**
AUC ROC score is 0.9170185930893853
[[11224   101]
 [  593  3183]]


----------------------------------------

**Processing cat__2 titles...**
AUC ROC score is 0.8913147383432121
[[13413    38]
 [  354  1296]]


----------------------------------------

**Processing cat__3 titles...**
AUC ROC score is 0.9184850294529227
[[13412   142]
 [  236  1311]]


----------------------------------------

**Processing cat__4 titles...**
AUC ROC score is 0.9090818059696308
[[13118    91]
 [  331  1561]]


----------------------------------------
Test averaged AUC ROC is 0.9633666644593074
```

Multinomial Naïve Bayes

In [25]:

```
# Multinomial Naive Bayes

NB_pipeline = Pipeline([('clf', OneVsRestClassifier(MultinomialNB(fit_prior=True, clas
accuracy = 0
for category in cat_col_names:
    print('**Processing {} titles...**'.format(category))

    # Training logistic regression model on train data
    NB_pipeline.fit(X_train, y_train[category])

    # calculating test accuracy
    prediction = NB_pipeline.predict(X_test)
    accuracy = accuracy + accuracy_score(y_test[category], prediction)
    print('AUC ROC is {}'.format(roc_auc_score(y_test[category],prediction)))
    conf_mat = confusion_matrix(
        y_test[category], prediction
```

```
        )
    print(conf_mat)
    print("\n")
    print("-"*40)


print('Test averaged AUC ROC is {}'.format(accuracy/len(cat_col_names)))
```

```
**Processing cat__0 titles...**
AUC ROC is 0.6598610057425418
[[8832   33]
 [4219 2017]]


----------------------------------------
**Processing cat__1 titles...**
AUC ROC is 0.5805084745762712
[[11325     0]
 [ 3168   608]]


----------------------------------------
**Processing cat__2 titles...**
AUC ROC is 0.5
[[13451     0]
 [ 1650     0]]


----------------------------------------
**Processing cat__3 titles...**
AUC ROC is 0.5
[[13554     0]
 [ 1547     0]]


----------------------------------------
**Processing cat__4 titles...**
AUC ROC is 0.5
[[13209     0]
 [ 1892     0]]


----------------------------------------
Test averaged AUC ROC is 0.8343288523938813
```

Classification Modeling Summary

Moving forward, we recommend using the Logistic Regression model for classification prediction for the prediction of our current NMF Topics. If the topics change in the future, then classification modeling needs to be reassessed. If better accuracy is needed, we suggest two additional things to try. One, model cut offs could be adjusted to see if there is improvement. Two, other models could be tested, like a neural network, could be tested to see if we can gain any improvement.
