Homework 2

> The body grows stronger under stress. The mind does not.
>
> -- Magic the Gathering, *Fractured Sanity*

Setup

Packages

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

Data

Run `dvc pull` to ensure my local copy of the repo has the actual data

In [2]:
```
!dvc pull
```

```
Everything is up to date.
ERROR: failed to pull data from the cloud - config file error: no remote sp
ecified. Create a default remote with
    dvc remote add -d <remote name> <remote url>
```

Load the data using pandas+pyarrow:

In [3]:
```
df = pd.read_feather('../../../data/mtg.feather')
df.head()[['name','text', 'mana_cost', 'flavor_text','release_date', 'edhrec_ra
```

Out[3]:

	name	text	mana_cost	flavor_text	release_date	edhrec_rank
0	Ancestor's Chosen	First strike (This creature deals combat damag...	[5, W, W]	<NA>	2007-07-13	16916.0
1	Angel of Mercy	Flying When Angel of Mercy enters the battlefi...	[4, W]	Every tear shed is a drop of immortality.	2007-07-13	14430.0
2	Aven Cloudchaser	Flying (This creature can't be blocked except ...	[3, W]	<NA>	2007-07-13	13098.0
3	Ballista Squad	{X}{W}, {T}: Ballista Squad deals X damage to ...	[3, W]	The perfect antidote for a tightly packed form...	2007-07-13	14972.0
4	Bandage	Prevent the next 1 damage that would be dealt ...	[W]	Life is measured in inches. To a healer, every...	2007-07-13	4980.0

Submission Structure

You will need to submit a pull-request on DagsHub with the following additions:

- your subfolder, e.g. named with your user id, inside the `homework/hw2-goals-approaches/` folder
  - your "lab notebook", as an `.ipynb` or `.md` (e.g. jupytext), that will be exported to PDF for Canvas submission. **This communicates your *goals***, along with the results that will be compared to them.
  - your `dvc.yaml` file that will define the inputs and outputs of your *approaches*. See the DVC documentation for information!
  - **source code** and **scripts** that define the preprocessing and prediction `Pipeline`'s you wish to create. You may then *print* the content of those scripts at the end of your notebook e.g. as appendices using
- any updates to `environment.yml` to add the dependencies you want to use for this homework

Part 1: Unsupervised Exploration

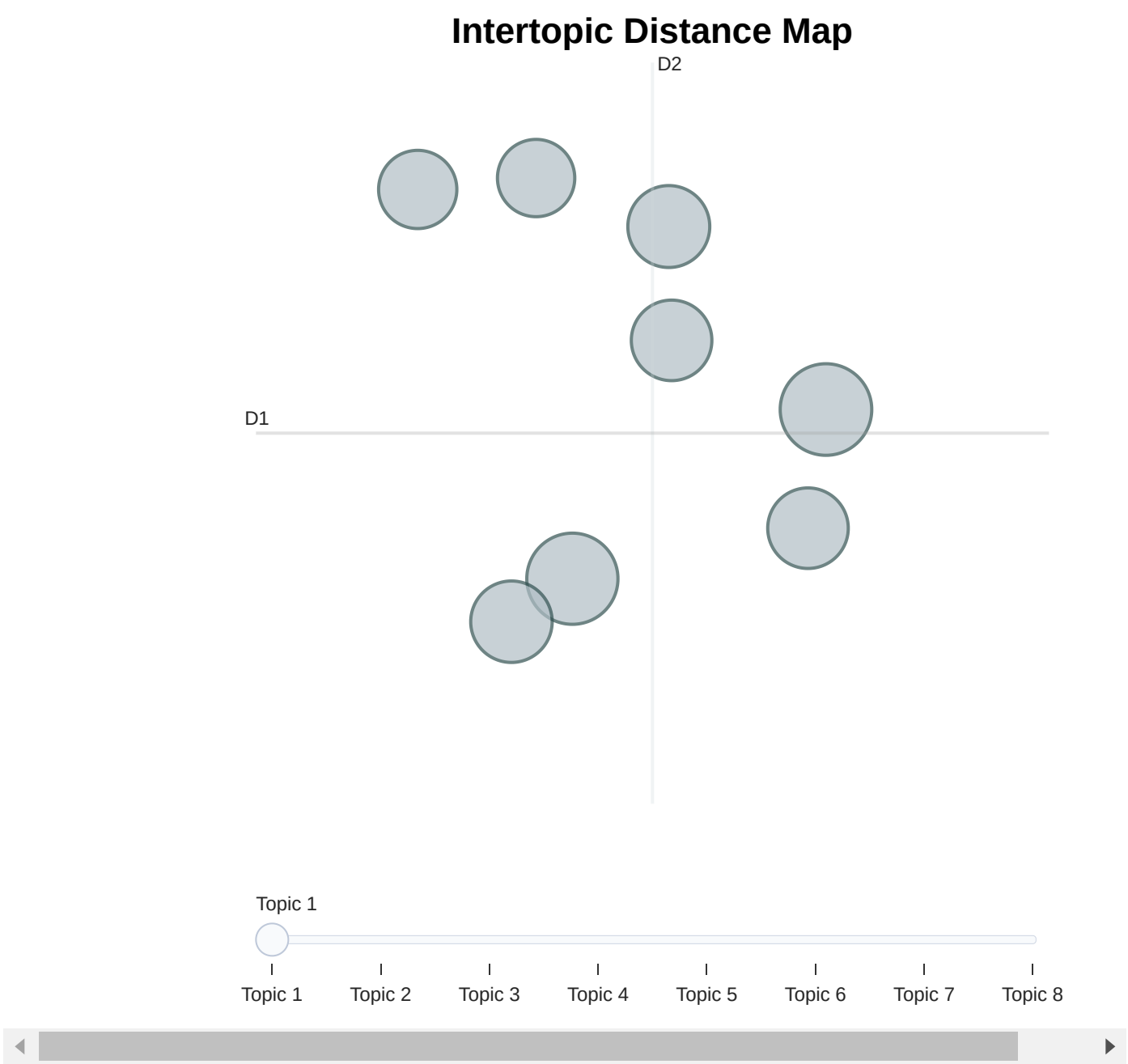
Investigate the BERTopic documentation (linked), and train a model using their library to create a topic model of the `flavor_text` data in the dataset above.

- In a `topic_model.py`, load the data and train a bertopic model. You will `save` the model in that script as a new trained model object
- add a "topic-model" stage to your `dvc.yaml` that has `mtg.feather` and `topic_model.py` as dependencies, and your trained model as an output
- load the trained bertopic model into your notebook and display
  1. the `topic_visualization` interactive plot see docs
  2. Use the plot to come up with working "names" for each major topic, adjusting the *number* of topics as necessary to make things more useful.
  3. Once you have names, create a *Dynamic Topic Model* by following their documentation. Use the `release_date` column as timestamps.
  4. Describe what you see, and any possible issues with the topic models BERTopic has created. **This is the hardest part... interpreting!**

In [4]:
```
from bertopic import BERTopic
```

In [5]:
```
# Load the BERTopic model
ft_model = BERTopic.load("ft_model")
```

In [6]:
```
# Visualize topics
ft_model.visualize_topics(top_n_topics = 9)
```



In [7]:
```
# Get topic information
ft_model.get_topic_info()[2:10].set_index("Topic")
```

Out[7]:

Topic	Count	Name
1	115	1_kami_kamigawa_observations_war
2	114	2_goblins_goblin_rivaled_innovations
3	92	3_jace_beleren_iquati_pol
4	91	4_hunt_hunters_lacerators_hunting
5	89	5_faeries_faerie_fae_oona
6	89	6_guild_guilds_guildless_guildpact
7	84	7_shadows_daylight_meld_clearly
8	82	8_necromancer_limdl_leshrac_recruitment

Renaming Topics

Topic 1: kami

Topic 2: goblin

Topic 3: wildspeaker

Topic 4: dragon

Topic 5: darkness

Topic 6: gerrard

Topic 7: mage

Topic 8: temper

In [8]:
```
# Prepare data
df_notna = df.dropna(subset=["flavor_text", "release_date"])
docs = df_notna.flavor_text.to_list()
timestamps = df_notna.release_date.to_list()
```

In [9]:
```
# Train a BERTopic model
topic_model = BERTopic(verbose=True)
topics, probs = topic_model.fit_transform(docs)

topics_over_time = topic_model.topics_over_time(docs, topics, timestamps)
```

```
Batches:   0%|          | 0/927 [00:00<?, ?it/s]
2022-05-03 09:36:03,305 - BERTopic - Transformed documents to Embeddings
2022-05-03 09:36:27,806 - BERTopic - Reduced dimensionality with UMAP
```

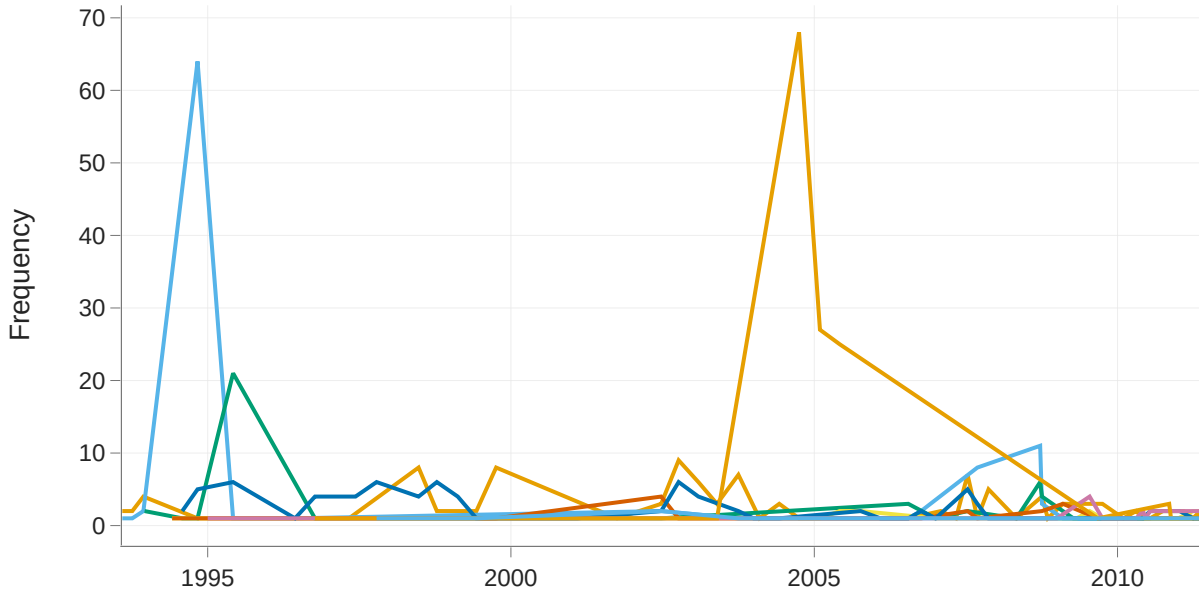
```
huggingface/tokenizers: The current process just got forked, after parallel
ism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(t
rue | false)
huggingface/tokenizers: The current process just got forked, after parallel
ism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(t
rue | false)
huggingface/tokenizers: The current process just got forked, after parallel
ism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(t
rue | false)
huggingface/tokenizers: The current process just got forked, after parallel
ism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(t
rue | false)
huggingface/tokenizers: The current process just got forked, after parallel
ism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(t
rue | false)
huggingface/tokenizers: The current process just got forked, after parallel
ism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(t
rue | false)
```

```
2022-05-03 09:36:30,532 - BERTopic - Clustered UMAP embeddings with HDBSCAN
281it [00:14, 19.13it/s]
```

In [10]:

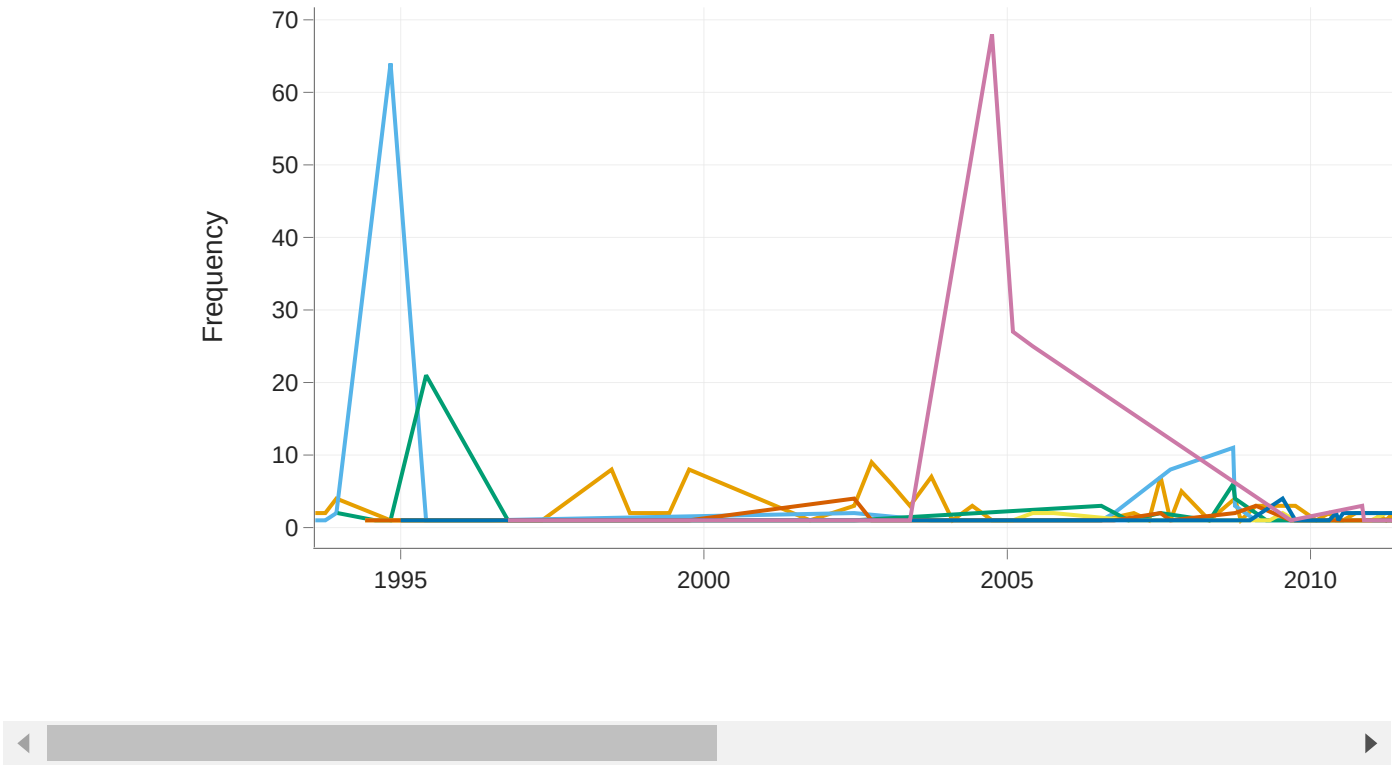
```python
# Visualize topics over time
topic_model.visualize_topics_over_time(topics_over_time, top_n_topics=9)
```

Topics over Time



Kami is the Japanese word for "great spirits." From the figure, we can see that the kami topic peaks around 2004, suggesting that there was probably an expansion set about kami released in 2004. After some research, I found out that *Champions of Kamigawa* was released in October 2004 as the first set in the Kamigawa block and it introduced many rare creatures with kami-like magics. I actually started playing mtg recently, and bought the *Kamigawa: Neon Dynasty* expansion set (released in February 2022) over the weekend. According to the storyline, this set represents the current era on Kamigawa, which is more than 1200 years after conclusion of the original Kamigawa block. If we excluded the 2004 kami outlier, we would probably see another smaller spike in 2022.

In [11]:
```
# Visualize topics over time without the 'kami' outlier
topic_model.visualize_topics_over_time(topics_over_time, topics=[0,1,3,4,5,6,7
```



After removing the kami outlier from the figure, we can see that the goblin topic peaks arouund 2012, 2016 and 2022, suggesting that goblin cards probably have gained popularity, resulting in multiple goblin-related expansion sets released over the years. This makes great sense to me, because I think that goblin cards are pretty strong and personally love using them.

Part 2 Supervised Classification

Using only the `text` and `flavor_text` data, predict the color identity of cards:

Follow the sklearn documentation covered in class on text data and Pipelines to create a classifier that predicts which of the colors a card is identified as. You will need to preprocess the target _ `color_identity` _ labels depending on the task:

- Source code for pipelines
  - in `multiclass.py`, again load data and train a Pipeline that preprocesses the data and trains a multiclass classifier (`LinearSVC`), and saves the model pickel output once trained. target labels with more than one color should be *unlabeled*!
  - in `multilabel.py`, do the same, but with a multilabel model (e.g. here). You should now use the original `color_identity` data as-is, with special attention to the multi-color cards.
- in `dvc.yaml`, add these as stages to take the data and scripts as input, with the trained/saved models as output.

- in your notebook:
    - Describe: preprocessing steps (the tokenization done, the ngram_range, etc.), and why.
    - load both models and plot the *confusion matrix* for each model (see here for the multilabel-specific version)
    - Describe: what are the models succeeding at? Where are they struggling? How do you propose addressing these weaknesses next time?

Multiclass

In [25]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
import pickle
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
```

In [13]:
```
# Load the multiclass model
multiclass_model = pickle.load(open("multiclass.sav", 'rb'))
```

Preprocessing:

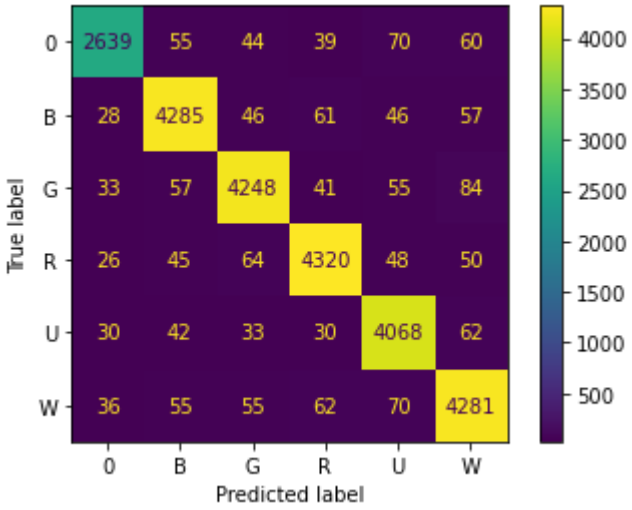
- Remove NAs in column _flavor*text*, *text*, and _color*identity*
- Remove multicolored _color*identity*
- min_df = 5, ignore rare words (appear in less than 5 documents)
- max_df = 0.8, ignore common words (appear in more than 80% of documents)

In [14]:
```
# Load the data
df = pd.read_feather("../../../data/mtg.feather")

# Remove NAs
df = df.dropna(subset = ["flavor_text", "text", "color_identity"]).reset_index

# Remove multicolored cards
df = df[df.color_identity.map(len) < 2]
```

In [15]:
```
# X
tfidf = TfidfVectorizer(
    min_df=5, # ignore rare words (appear in less than 5 documents)
    max_df=0.8, # ignore common words (appear in more than 80% of documents)
    stop_words="english"
)
text = df["text"] + df["flavor_text"].fillna('')
X = tfidf.fit_transform(text)

# y
ci = [list(i)[0] if len(i) == 1 else 0 for i in df.color_identity]
le = preprocessing.LabelEncoder()
y = le.fit_transform(ci)
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=111)
```

In [27]:
```
ConfusionMatrixDisplay.from_estimator(multiclass_model, X, y, display_labels=le
# plot_confusion_matrix(multiclass_model, X_test, y_test)
plt.show()
```



Multilabel

In [28]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import multilabel_confusion_matrix
```

In [42]:
```
# Load the multilabel model
multilabel_model = pickle.load(open("multilabel.sav", 'rb'))
```

Preprocessing:

- min_df = 5, ignore rare words (appear in less than 5 documents)
- max_df = 0.8, ignore common words (appear in more than 80% of documents)
- Transform _color*identity* into lowecase

In [43]:
```
# Instantiate OneVsRestClassifier
# multilabel_model = OneVsRestClassifier(SVC(kernel="linear"))

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=111)

# Fit OneVsRestClassifier
# multilabel_model.fit(X_train, y_train)
```

In [44]:
```
# Save the model using pickle
# pickle.dump(multilabel_model, open("multilabel.sav", 'wb'))
```

In [45]:
```
# Load the data
df = pd.read_feather("../../../data/mtg.feather")
```

In [46]:
```
# X
tfidf = TfidfVectorizer(
    min_df=5, # ignore rare words (appear in less than 5 documents)
    max_df=0.8, # ignore common words (appear in more than 80% of documents)
    stop_words="english"
)
text = df["text"] + df["flavor_text"].fillna('')
X = tfidf.fit_transform(text)
```

In [47]:
```
# y
ci = df["color_identity"]
cv = CountVectorizer(tokenizer=lambda x: x, lowercase=False)
y = cv.fit_transform(ci)
```

In [48]:
```
y_pred = multilabel_model.predict(X_test)
multilabel_confusion_matrix(y_test, y_pred)
```

Out[48]:
```
array([[[10651,   330],
        [ 1011,  2100]],

       [[10729,   267],
        [ 1127,  1969]],

       [[10802,   259],
        [  975,  2056]],

       [[10641,   390],
        [ 1010,  2051]],

       [[10716,   337],
        [ 1057,  1982]]])
```

The multiclass model performs well for certain labels but not all of them. However, the multilabel model reaches an overall high accuracy.

Part 3: Regression?

> Can we predict the EDHREC "rank" of the card using the data we have available?

- Like above, add a script and dvc stage to create and train your model.
- In the notebook, aside from your descriptions, plot the `predicted` vs. `actual` rank, with a 45-deg line showing what "perfect prediction" should look like.
- This is a freeform part, so think about the big picture and keep track of your decisions:
  - what model did you choose? Why?
  - What data did you use from the original dataset? How did you proprocess it?
  - Can we see the importance of those features? e.g. logistic weights?

How did you do? What would you like to try if you had more time?

In [49]:
```
from sklearn.linear_model import LinearRegression, Lasso
```

I choose to perform a regression model, and feed in the following as X:

- converted_mana_cost
- power
- toughness
- color_identity (create a dummy variable for each color)
- keywords (create a dummy variable of whether the card has any keywords)
- rarity (create a dummy variable for each rarity level)

In [50]:
```
# Load the regression models
linear_model = pickle.load(open("linear.sav", 'rb'))
lasso_model = pickle.load(open("lasso.sav", 'rb'))
```

In [51]:
```
# Load the data
df = pd.read_feather("../../../data/mtg.feather")

# Remove NAs in column edhrec_rank
df = df.dropna(subset = ["edhrec_rank"]).reset_index(drop=True)
```

In [52]:
```
# converted_mana_cost
# convert to int
df["converted_mana_cost"] = df["converted_mana_cost"].astype(int)
```

In [53]:
```
# power
# Replace NaN power with 0
df["power"] = df["power"].fillna(0).astype(int)

# toughness
# Replace NaN toughness with 0
df["toughness"] = df["toughness"].fillna(0).astype(int)
```

In [54]:
```
# color_identity
# Create a df for color_identity dummy variables
colors = pd.get_dummies(df.color_identity.apply(pd.Series).stack()).sum(level=0
# Merge df with colors on index
df = df.join(colors)
# Replace NaN color_identity dummy variables with 0
df[colors.columns] = df[colors.columns].fillna(0).astype(int)
```

In [55]:
```
# keywords
# no_keywords = 0 & yes_keywords = 1
df["keywords"] = df["keywords"].isnull().astype(int)
```

In [56]:
```
# rarity
# Create a dummy variable for each rarity level
df = pd.get_dummies(df, columns=["rarity"])
```

In [57]:
```
df = df[['converted_mana_cost', 'power', 'toughness',
         'B', 'G', 'R', 'U', 'W', 'keywords',
         'rarity_rare', 'rarity_common', 'rarity_uncommon',
         'edhrec_rank']]
```

In [58]:
```
y = df['edhrec_rank']
X = df.drop('edhrec_rank', axis=1)
```

In [59]:
```
# Train-test split
```

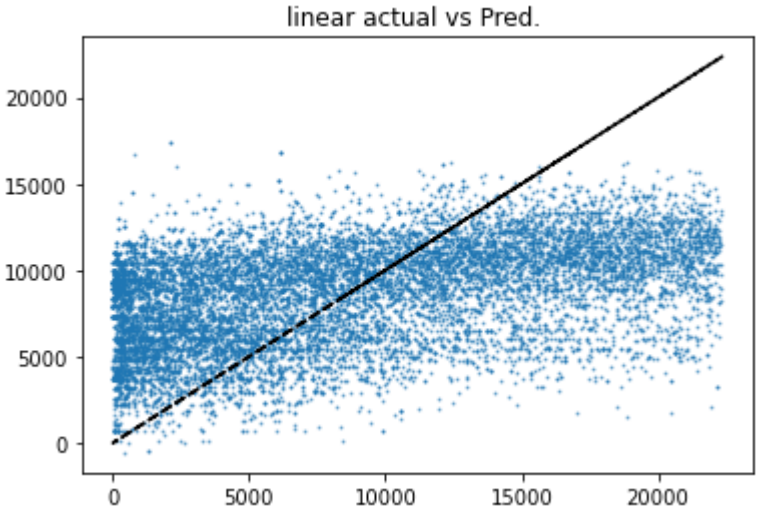
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=111)
```

In [60]:
```
# LinearRegression score
linear_model.score(X_test, y_test)
```

Out[60]:
```
0.20249130421440364
```

In [74]:
```
plt.plot(y,y, color='k', ls='--')
plt.scatter(y_test, linear_model.predict(X_test), alpha=0.5, s=1)

plt.title('linear actual vs Pred. ')
```

Out[74]:
```
Text(0.5, 1.0, 'linear actual vs Pred. ')
```

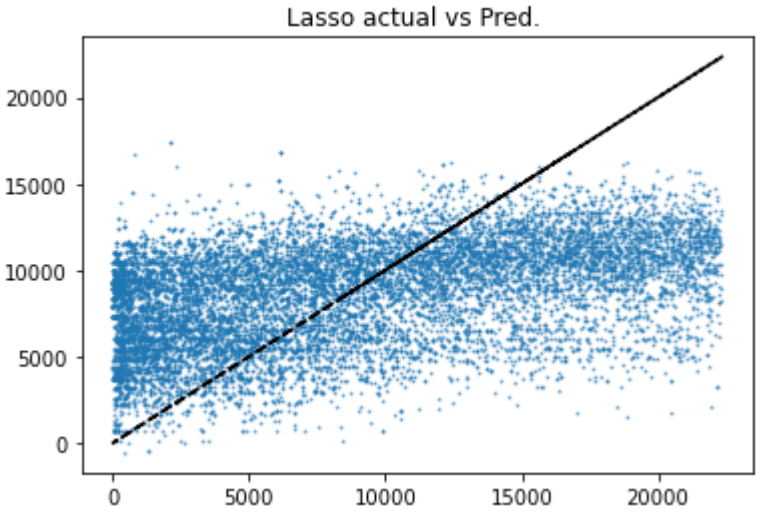


In [61]:
```
# Lasso score
lasso_model.score(X_test, y_test)
```

Out[61]:
```
0.20249130421440364
```

In [72]:
```
plt.plot(y,y, color='k', ls='--')
plt.scatter(y_test, lasso_model.predict(X_test), alpha=0.5, s=1)

plt.title('Lasso actual vs Pred. ')
```

Out[72]:
```
Text(0.5, 1.0, 'Lasso actual vs Pred. ')
```



In [ ]: