Homework 2/3

> 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.
```

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

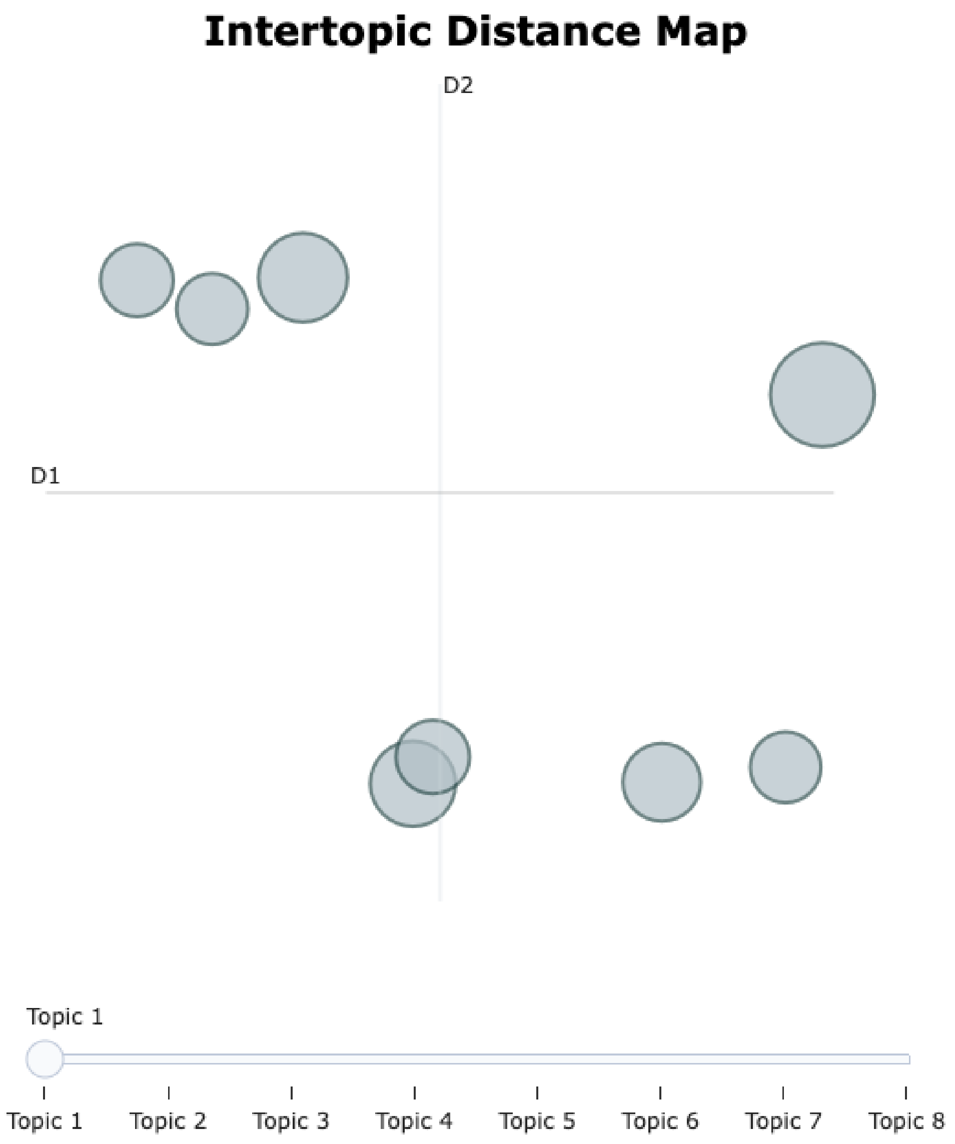
Investigate the BERTopic documentation (linked), and train a model using their library to create a topic model of the `flavor_text` data in the dataset above.

- 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 [8]:
```
# Get topic information
ft_model.get_topic_info()[2:10].set_index("Topic")
```

Out[8]:

	Count	Name
Topic		
1	185	1_kami_kamigawa_observations_konda
2	136	2_goblin_cabin_goblins_squee
3	125	3_wildspeaker_garruk_gavi_geor
4	104	4_dragons_dragon_caustic_digest
5	93	5_shadows_fragrance_violet_pauses
6	91	6_gerrard_gerrards_legacy_orim
7	87	7_mages_mage_emulate_retinues
8	86	8_tempers_geyser_refract_dizzying

Renaming Topics

Topic 1: kami
Kami is the Japanese word for "great spirits" and kamigawa means "river of the gods." This topic contains all things about gods and their world.

Topic 2: goblin
Goblin is a type of creature in mtg and Squee is a quite common name for goblin. This topic is about the goblin creature.

Topic 3: Garruk Wildspeaker
Garruk Wildspeaker is a planewalker. Since planewalkers are considered the most powerful beings in mtg, many flavor texts may refer to them.

Topic 4: dragon
Dragon is a type of creature in mtg.

Topic 5: darkness
Many of the mtg stories would mention dark magic of wizards, skeletons, etc. This topic is probably all about the dark side.

Topic 6: gerrard
Gerrard Capashen is the protagonist of the mtg storyline. The "legacy" word appears in this topic because Garrard is the heir to Urza, a famous planewalker. This topic is telling the story around Garrard.

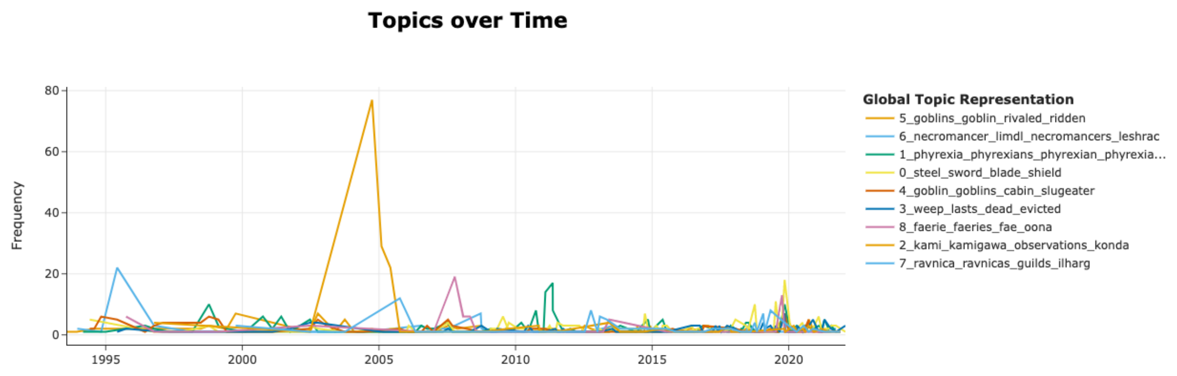
Topic 7: mage
Since mtg is all about magic, it makes sense that mage would appear in various flavor texts.

Topic 8: instant/socery
After some research, I found that temper (Fiery Temper), geyser (Mana Geyser) and refract (refraction trap) are all among the instant or sorcery cards.

In [9]:
```
# Prepare data
df_notna = df.dropna(subset=["flavor_text", "release_date"]).reset_index(drop=T
docs = df_notna.flavor_text.to_list()
timestamps = df_notna.release_date.to_list()
```

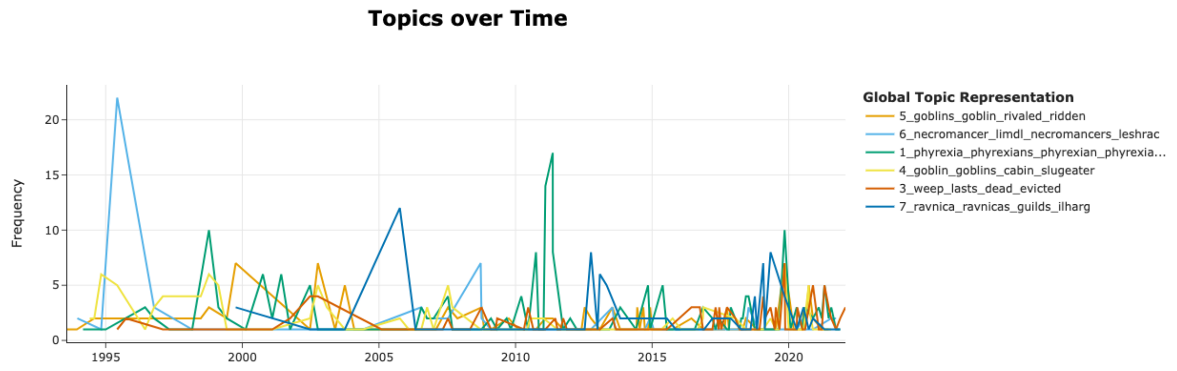
In [10]:
```
topics, probs = ft_model.fit_transform(docs)
topics_over_time = ft_model.topics_over_time(docs, topics, timestamps)
```

In [11]:
```
# Visualize topics over time
ft_model.visualize_topics_over_time(topics_over_time, top_n_topics=9)
```



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 [12]:
```python
# Visualize topics over time without the 'kami' outlier
ft_model.visualize_topics_over_time(topics_over_time, topics=[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 [13]:
```
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
```

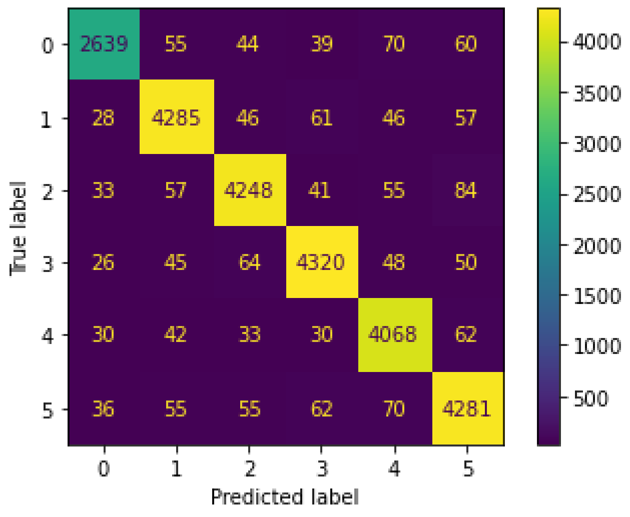
In [14]:
```
# Load the multiclass model
multiclass_model = pickle.load(open("multiclass.sav", 'rb'))
```

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 [15]:
```
from multiclass import X, y
```

In [16]:
```
ConfusionMatrixDisplay.from_estimator(multiclass_model, X, y)
plt.show()
```



Multilabel

In [17]:
```
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 [18]:
```
# 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 [19]:
```
from multilabel import X_test, y_test
```

In [50]:
```
y_pred = multilabel_model.predict(X_test)
multilabel_confusion_matrix(y_test, y_pred)
```

Out[50]:
```
array([[[10651,   330],
        [ 1011,  2100]],

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

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

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

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

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 [20]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.inspection import permutation_importance
from plotnine import *
```

In [21]:
```
# Load the regression model
reg_model = pickle.load(open("regression.sav", 'rb'))
```

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

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

In [23]:
```
from regression import X, y, X_train, y_train, X_test, y_test
```

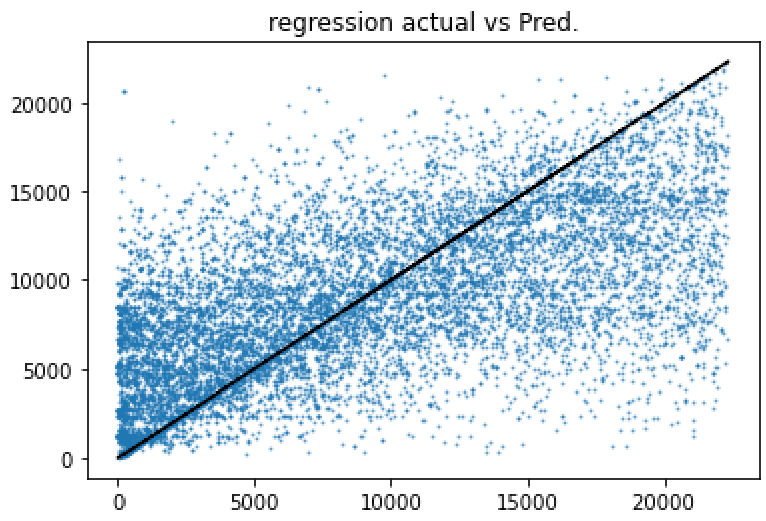
In [24]:
```
# Regression model score
reg_model.score(X_test, y_test)
```

Out[24]:
```
0.4129678188999356
```

In [28]:
```
plt.plot(y, y, color='k', ls='--')
plt.scatter(y_test, reg_model.predict(X_test), alpha=0.5, s=1)

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

Out[28]:
```
Text(0.5, 1.0, 'regression actual vs Pred.')
```



Extra Credit (5 pts)

In [31]:
```python
vi = permutation_importance(reg_model, X_train, y_train, n_repeats=5)
```

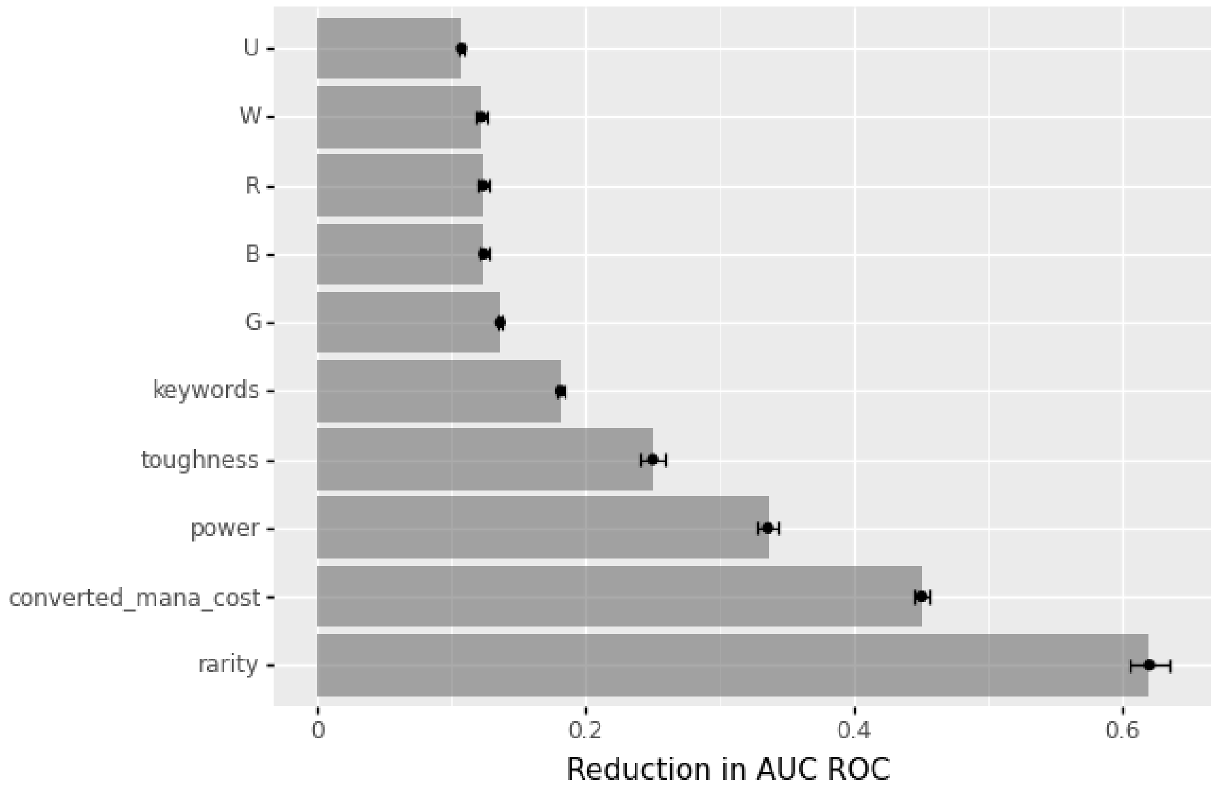
In [32]:
```python
# Organize as a df
vi_df = pd.DataFrame(dict(variable=X_train.columns,
                          vi = vi['importances_mean'],
                          std = vi['importances_std']))

# Generate intervals
vi_df['low'] = vi_df['vi'] - 2*vi_df['std']
vi_df['high'] = vi_df['vi'] + 2*vi_df['std']

# Put in order from most to least important
vi_df = vi_df.sort_values(by="vi", ascending=False).reset_index(drop=True)
```

In [35]:
```python
top10 = vi_df[0:10]

# Plot
(
    ggplot(top10,
           aes(x="variable",y="vi")) +
    geom_col(alpha=.5) +
    geom_point() +
    geom_errorbar(aes(ymin="low", ymax="high"), width=.2) +
    scale_x_discrete(limits=top10.variable.tolist()) +
    coord_flip() +
    labs(y="Reduction in AUC ROC",x="")
)
```



Out[35]:
```
<ggplot: (337006577)>
```

From the variable importance plot, we can see that `rarity` is the most important of all variables. It makes sense, because the rarer a card is, the higher it ranks. `converted_mana_cost` ranks the second, suggesting that the more mana a card needs, the stronger. `power` and `toughness` play a big part because they basically represent how strong a card attacks and blocks. The `keywords` variable here is a dummy on whether the card has keywords. In mtg, if a card has some keywords, it usually means that it has extra abilities, such as flying, haste, etc. It seems that the `color-identity` of a card doesn't matter that much, which is understandable, because there are strong cards in every color.

Part 4: Iteration, Measurement, & Validation

> No model is perfect, and experimentation is key. How can we more easily iterate and validate our model?

- Pick **ONE** of your models above (regression, multilabel, or multiclass) that you want to improve or investigate, and calculate metrics of interest for them to go beyond our confusion matrix/predicted-actual plots:
  - for multiclass, report average and F1
  - for multilabel, report an appropriate metric (e.g. `ranking_loss`)
  - for regression, report an appropriate metric (e.g. 'MAPE' or MSE), **OR** since these are ranks, the pearson correlation between predicted and actual may be more appropriate?

- in the corresponding `dvc.yaml` stage for your model-of-interest, add `params` and `metrics`
  - under `params`, pick a setting in your preprocessing (e.g. the `TfidfVecorizer`) that you want to change to imrpove your results. Set that param to your current setting, and have your model read from a `params.yaml` rather than directly writing it in your code.
  - under `metrics`, reference your `metrics.json` and have your code write the results as json to that file, rather than simply printing them or reporting them in the notebook.
- commit your changes to your branch, run `dvc repro dvc.yaml` for your file, then run a new experiment that changes that one parameter: e.g. `dvc exp run -S preprocessing.ngrams.largest=1` (see the `example/` folder for a complete working example).

Report the improvement/reduction in performance with the parameter change for your metric, whether by copy-pasting or using `!dvc exp diff` in the notebook, the results of `dvc exp diff`.

In [39]:
```
from multiclass_pipe import X_train, X_test, y_train, y_test
from sklearn.metrics import classification_report
```

In [40]:
```
# Load the multiclass_pipe model
multiclass_pipe = pickle.load(open("multiclass_pipe.sav", 'rb'))
```

In [43]:
```
y_pred = multiclass_pipe.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           B       0.80      0.76      0.78       552
           G       0.80      0.77      0.79       524
           R       0.80      0.80      0.80       506
           U       0.77      0.85      0.81       505
           W       0.74      0.74      0.74       549

    accuracy                           0.78      2636
   macro avg       0.78      0.78      0.78      2636
weighted avg       0.78      0.78      0.78      2636
```

In [49]:
```
# dvc exp run -S preprocessing.ngrams.largest=3
!dvc exp diff
```

```
Path          Metric     HEAD    workspace    Change
metrics.json  f1-score   -       0.78583      diff not supported
metrics.json  precision  -       0.78648      diff not supported
metrics.json  recall     -       0.78604      diff not supported
metrics.json  support    -       2636         diff not supported

Path         Param                           HEAD    workspace    Change
params.yaml  preprocessing.ngrams.largest    -       3            diff not supp
orted
params.yaml  preprocessing.ngrams.smallest   -       1            diff not supp
orted
```