Finding Strong MTG Decks

Created: 01/07/19 by Tom Lever
Updated: 04/26/19 by Tom Lever

Abstract

Magic the Gathering is a beautiful and complex trading card game, for children and adults alike, based on casting spells and either outlasting or defeating one or more opponents. There is a definite mystical and nostalgic atmosphere to the game, and in fact Magic cards are designed to share mechanics and flavors that reflect stories. Magic the Gathering Arena, an internet-based gameplay environment released in Fall 2018, makes playing Magic accessible and captivating, and does an excellent job performing and animating a vast number of intricately related actions. Given how complex playing Magic is and, by extension, how difficult it is to create strong Magic decks, I wrote software that creates strong decks based on gatherer.wizards.com's cards database, mtgarena.pro's deck database, and my own rules-text categories database. In this paper I present my software process and results. A strong deck list involving "Forest" and “Plains” cards generated by my software won 2 out of 3 games in the “Ranked” mode of MTG Arena at "Gold Tier 3".

1. Motivation

At the most basic level, a game of Magic the Gathering involves two players. Each player begins the game with twenty life points. Each player shuffles their own library of sixty cards and draws a seven-card hand. Players alternate in taking turns. A player's turn consists of a beginning phase, a pre-combat main phase, a combat phase, a post-combat main phase, and an ending phase. Each phase consists of one or more steps. During a main phase of the active player (i.e., the player whose turn it is), the active player may convert a land card in their hand into a land permanent on the battlefield under their control. During any phase, under certain conditions, a player may tap land permanents for the rest of their turn to create mana that can be used within the present step in a process of casting spells. Some spells cast by the active player, usually cast during a main phase of the active player, result in the creation of creature permanents on the battlefield under the caster's control. During the active player's combat phase, the active player often taps one or more creatures to have them attack the other player. During the active player's beginning phase, the active player usually resets tapped permanents to an untapped state and draws a card. During the ending phase, the game automatically performs some actions.

What makes Magic such an engaging game is how players and their permanents interact. On a basic level, players can cast some spells, activate some abilities of permanents, and take some other actions, like blocking attacks, during others' turns. On a more interesting level, players have many opportunities during each other's turns to cast spells and activate abilities to negate attacks, strengthen their own creatures or incapacitate others' creatures, better themselves and harm their opponents, and prepare for future turns. On an advanced level, some permanents even have abilities that are triggered automatically when certain events occur, which then allow for other abilities and spells to be activated, triggered, or cast, often resulting in cascades of abilities and spells. The number of interactions, number of cards that may be included in libraries, and the memorable flavors of Magic cards make for complex and beautiful gameplay that rather well imitates what I believe to be a widely held, romantic image of mage battles.

The complexity of Magic the Gathering, while rewarding and captivating, makes learning to play Magic and constructing libraries rather difficult. In this paper, I present software to aid in creating strong decks. My software should be of interest to any regular Magic player who has ever been frustrated by the complexity and open-endedness that I believe is inherent in constructing Magic decks.

2. Datasets

My software for developing strong MTG decks relies on three main datasets. The first is a database of information on 1,617 cards in the Ixalan, Rivals of Ixalan, Dominaria, Core Set 2019, Guilds of Ravnica, and Ravnica Allegiance card sets that I created using information from https://gatherer.wizards.com/ (https://gatherer.wizards.com/).

My second main dataset is a database of card names, win / loss ratios, and card counts associated with 2,271 proven decks (proven by virtue of having win / loss ratios) ensured to have sixty cards. I created this database using information from https://mtgarena.pro/decks/?community/ (https://mtgarena.pro/decks/?community/).

My third main dataset is my own rules-text categories database, which classifies each card in a condensed version of the cards database into one or more of fifty categories based on the content of that card's rules text.

3. Data Preparation and Cleaning

To prepare my cards database for analysis, I created a condensed cards database by deleting rows from my cards database with the same name, except the first row. The clearest application of this is aggregating basic land cards found in multiple card sets into one basic land card in one set. This aggregation was an automatic process. The condensed database has 1,546 cards. I also manually adjusted the mana types of lands from being all "Colorless" to being "Colorless", associated with one specific mana type, associated with two mana types, or associated with any mana type. The rarities for the five basic land cards is “Land” so that the basic lands appear at the top of the best-cards database when the best-cards database is sorted first by rarity and then by total number of occurrences. The condensed cards database is used first to create a best-cards database.

Constructing a name x ratio database required locating information on 4,827 decks, then scraping together information on 2,412 proven decks, then extracting information on 2,271 decks ensured to have sixty cards, then creating a table of mtgarena.pro card ID numbers and card names in those 2,271 decks, then creating a ID / name x location / win ratio database filled in with card counts, then aggregating rows by name in and eliminating the ID column and location row from the ID / name x location / win ratio database. Creating the name x ratio database took a lot of prep work and it was a little difficult to reload this table once saved to a CSV file due to duplicate column headers.

4. Research Question

The fundamental research question that I answered using my software was, “What is a strong deck of sixty nonland cards where all the cards:

1. Are in my MTG-Arena inventory;
2. Have mana costs in a certain grouping [e.g., (“C”, “F”, "P", "CF", "CP", "FP", “CFP”)];
3. Each belong to one or more categories in a grouping of categories associated with a high win / loss ratio [e.g., (“create creature token or convert lands into creatures”, "destroy", "draw", "put nonland card", "flying")];
4. Are balanced so that more important categories have higher proportions of cards; and
5. Have a maximum average converted mana cost less than a certain user-defined maximum [e.g., 3.001]?"

5. Summary of Methods

My software, executed one module at a time, completes the tasks in the below checklist.

1. Condense cards database by deleting rows with the same name, except the first row. The clearest application of this is aggregating basic land cards found in multiple card sets into one basic land card in one set. My condensed database has 1,546 cards. The condensed cards database will be used first to create a best-cards database. After condensing the cards database, I manually adjusted the mana types of lands from being all "Colorless" to being "Colorless", associated with one mana type, associated with two mana types, or associated with any mana type.

2. Create a raw table of deck URL's and any win ratios. The URL's are to information on 4,827 proven or unproven decks read from https://mtgarena.pro/decks/?community/. A URL may not have a corresponding win ratio. A cleaned table will be created using this raw table.

3. Create a cleaned table of URL's and win ratios. The URL's are to information on 2,412 proven decks read from https://mtgarena.pro/decks/?community/ and the win / loss ratios corresponding to those decks. The URL's will be used first to archive information on each deck to its own text file. The win / loss ratios will be used first to create a winnowed table of URL's and win / loss ratios for decks ensured to have sixty cards.

4. Archive information on each of 2,412 proven decks in its own text file in a folder of files of deck information. A similar archive will be created from this archive for information on each of 2,271 decks ensured to have sixty cards.

5. Archive information on sixty-card decks and create a winnowed cleaned table of URLs and ratios. The archived information will be used first to create a table of card ID numbers and names of cards in each deck ensured to have sixty cards. The winnowed cleaned table of URLs and ratios will be used first as the multirow header of a (card ID / card name) $\times$ (deck URL / win/loss ratio) database.

6. Create a table of card ID numbers and names. Each (ID number, name) pair will correspond to one card in at least one of the 2,271 decks ensured to have sixty cards. Each (ID number, name) pair in the table will be unique. The ID numbers and names for cards in each deck are found in the BeautifulSoup for that deck. Unfortunately, different ID numbers may have the same card name. I handle this by aggregating rows with the same card name in the (card ID / card name) $\times$ (deck URL / win/loss ratio) database. This table of ID numbers and names will be used first in the multicolumn index of the (card ID / card name) $\times$ (deck URL / win/loss ratio) database.

7. Create a (card ID / card name) $\times$ (deck URL / win/loss ratio) database. A condensed database without MTGArena.pro ID numbers, without deck-page URL's, and with rows aggregated by name will be created using this database.

8. Create a name × ratio database. MTGArena.pro ID numbers will be eliminated, URL's will be eliminated, and rows will be aggregated by name. The condensed database will be used first to create a table of card names, total numbers of occurrences, and average frequencies of cards in 2,271 decks.

9. Create a table of card names, total occurrences, and frequencies. The table will be used first to create a best-cards database.

10. Create a best-cards database. A best-cards database with an inventory column will be created using the best-cards database.

11. Create a best-cards database with inventory. The inventory column of this database will be filled in from an inventory database by an Excel VBA module. The best-cards database will be sorted first by rarity and second by total number of occurrences in 2,271 decks. This database will be filtered into strong decks.

12. Find average win ratios of decks with mana types in specific groupings. The groupings and average win ratios will be output to this notebook. To satisfy my desires to enjoy play and to play competitively, I will filter the best-cards database with inventory into a strong deck with mana types corresponding to a flavor that I enjoy playing and corresponding to a grouping with a high average win ratio.

13. Develop rules text categories database. My rules text categories database classifies every card in the condensed cards database as belonging to one or more of fifty categories based on the content of the card's rules text. A categories of interest database will be created using this database.

14. Develop categories of interest database. My categories of interest database classifies every card in the condensed cards database into one or more of half a dozen to a dozen categories of interest. When I only had half of the rules text categories database filled in, I used the below machine-learning techniques to fill in the second half of the appropriate categories of interest columns. When Wizards of the Coast publishes a new card set, I will be able to use the below machine-learning techniques to guess at how the new cards may be assigned to my fifty categories. Regardless of whether or not I am using my machine-learning program to fill in the categories of interest database, I will create a filled-in categories of interest database using the categories of interest database.

15. Fill in categories database by saving a copy of "Categories_of_Interest.csv" as "Filled_In_Categories_Database.csv" or by using a machine-learning program. A filtered filled-in categories database will be created using this database.

16. Filter filled in categories database by mana type and availability. This database will be used to find the highest average win ratios for all combinations of categories in the filled-in categories database.

17. Determine highest average win / loss ratios for combinations of categories. For each possible number of unique cards in a deck that each happen to be in at least one category in a grouping, the win / loss ratios of all decks with that number of unique cards in categories are averaged. The highest average

win / loss ratio assigned to a grouping of categories is the highest average win / loss ratio among average win / ratios for different numbers of unique cards in categories.

18. Weight categories in a chosen grouping.

19. Develop a strong deck list with specific mana types, rules-text categories, and maximum average converted mana cost. The mana types were specified in the program that filtered the filled-in categories database. The rules-text categories were specified in the program that developed the categories of interest database. The maximum average converted mana cost is defined in this program. I start of with an unrealistically high max ave CMC (i.e., 12). I vary the max ave CMC based on whether I feel that gameplay with the base unconstrained deck is too "slow" / "heavy". I am interested in whether constrained decks out-perform unconstrained decks. This strong deck list will be used to play enjoyable and competitive games.

6. Walking through Software Solutions to Checklist Tasks

6.1. Condensing Cards Database

Return to Tasks Checklist

A Magic card may be converted into a land permanent; used to cast instant and sorcery spells; or converted into an artifact, creature, artifact creature, enchantment, or planeswalker permanent. Each Magic cards has a name. Each nonland card has a mana cost, which indicates the number and types of land permanents that must be tapped to use that card. Each cards has a high-quality illustration taking up roughly half the card. Each card has a type, and many cards have subtypes. Each card has a set icon, which is filled with one of four colors to indicate rarity. Each instant or sorcery card has rules text that indicates spell effects. Each permanent card has rules text that indicates triggered abilities, or activated abilities and their costs. Each creature card has a power indicator and a toughness indicator, and each planeswalker card has a loyalty indicator.

I wrote the below Python program to create a condensed cards database to information on 1,546 cards in my cards database. Rows with the same card information except card set are aggregated. I would like to note that in my original cards database I initially set all lands to have a mana type of Colorless, given that their mana costs are 0. After condensing the cards database, I manually adjusted all land mana types in the condensed cards database to one of "C", "F", "I", "M", "P", "S", "FI", "FM", "FP", "FS", "IM", "IP", "IS", "MP", "MS", "PS", or "Any". Please see below screenshot of the condensed cards database.

My program requires the path to my initial cards database. My program relies on the pandas Python library for reading, aggregating, and writing. My program outputs the condensed cards database to "Condensed_Cards_Database.csv" in the "Data_With_Ravnica_Allegiance" subfolder to this notebook's folder.

Name	Mana Cost	Mana Type	Converted Mana Cost	Card Type	Subtypes	Set	Rarity	Power	Toughness	Loyalty	Rules Text
Adanto Vanguard	[1][White]	CP	2	Creature	Vampire Soldier	Ixalan	2	1	1		As long as Adanto Vanguard is attacking| it gets +2/+0.[LINE BREAK]Pay 4 life: Adanto Vanguard gains indestructible until end of turn. (Damage and effects that say "destroy" don't destroy it.)
Adanto| the First Fort	[0]	P	0	Legendary Land	None	Ixalan	3				(Transforms from Legion's Landing.)[LINE BREAK][Tap]: Add [White].[LINE BREAK][2][White]| [Tap]: Create a 1/1 white Vampire creature token with lifelink.
Admiral Beckett Brass	[1][Blue][Black][Red]	CISM	4	Legendary Creature	Human Pirate	Ixalan	4	3	3		Other Pirates you control get +1/+1.[LINE BREAK]At the beginning of your end step| gain control of target nonland permanent controlled by a player who was dealt combat damage by three or more Pirates this turn.
Air Elemental	[3][Blue][Blue]	CI	5	Creature	Elemental	Ixalan	2	4	4		Flying (This creature can't be blocked except by creatures with flying or reach.)
Ancient Brontodon	[6][Green][Green]	CF	8	Creature	Dinosaur	Ixalan	1	9	9		None
Angrath's Marauders	[5][Red][Red]	CM	7	Creature	Human Pirate	Ixalan	3	4	4		If a source you control would deal damage to a permanent or player| it deals double that damage to that permanent or player instead.
Anointed Deacon	[4][Black]	CS	5	Creature	Vampire Cleric	Ixalan	1	3	3		At the beginning of combat on your turn| you may have target Vampire get +2/+0 until end of turn.
Arcane Adaptation	[2][Blue]	CI	3	Enchantment	None	Ixalan	3				As Arcane Adaptation enters the battlefield| choose a creature type.[LINE BREAK]Creatures you control are the chosen type in addition to their other types. The same is true for creature spells you control and creature cards you own that aren't on the battlefield.
Arguel's Blood Fast	[1][Black]	CS	2	Legendary Enchantment	None	Ixalan	3				[1][Black]| Pay 2 life: Draw a card.[LINE BREAK]At the beginning of your upkeep| if you have 5 or less life| you may transform Arguel's
Ashes of the Abhorrent	[1][White]	CP	2	Enchantment	None	Ixalan	3				Players can't cast spells from graveyards or activate abilities of cards in graveyards.[LINE BREAK]Whenever a creature dies| you gain 1
Atzocan Archer	[2][Green]	CF	3	Creature	Human Archer	Ixalan	2	1	4		Reach[LINE BREAK]When Atzocan Archer enters the battlefield| you may have it fight another target creature. (Each deals damage equal to its power to the other.)
Axis of Mortality	[4][White][White]	CP	6	Enchantment	None	Ixalan	4				At the beginning of your upkeep| you may have two target players exchange life totals.
Azcanta| the Sunken Ruin	[0]	I	0	Legendary Land	None	Ixalan	3				(Transforms from Search for Azcanta.)[LINE BREAK][Tap]: Add [Blue].[LINE BREAK][2][Blue]| [Tap]: Look at the top four cards of your library. You may reveal a noncreature| nonland card from among them and put it into your hand. Put the rest on the bottom of your
Belligerent Brontodon	[5][Green][White]	CFP	7	Creature	Dinosaur	Ixalan	2	4	6		Each creature you control assigns combat damage equal to its toughness rather than its power.
Bellowing Aegisaur	[5][White]	CP	6	Creature	Dinosaur	Ixalan	2	3	5		Enrage â€" Whenever Bellowing Aegisaur is dealt damage| put a +1/+1 counter on each other creature you control.
Bishop of Rebirth	[3][White][White]	CP	5	Creature	Vampire Cleric	Ixalan	3	3	4		Vigilance[LINE BREAK]Whenever Bishop of Rebirth attacks| you may return target creature card with converted mana cost 3 or less from your graveyard to the battlefield.
Bishop of the Bloodstained	[3][Black][Black]	CS	5	Creature	Vampire Cleric	Ixalan	2	3	3		When Bishop of the Bloodstained enters the battlefield| target opponent loses 1 life for each Vampire you control.
Bishop's Soldier	[1][White]	CP	2	Creature	Vampire Soldier	Ixalan	1	2	2		Lifelink
Blight Keeper	[Black]	S	1	Creature	Bat Imp	Ixalan	1	1	1		Flying[LINE BREAK][7][Black]| [Tap]| Sacrifice Blight Keeper: Target opponent loses 4 life and you gain 4 life.
Blinding Fog	[2][Green]	CF	3	Instant	None	Ixalan	1				Prevent all damage that would be dealt to creatures this turn. Creatures you control gain hexproof until end of turn. (They can't be the targets of spells or abilities your opponents control.)
Bloodcrazed Paladin	[1][Black]	CS	2	Creature	Vampire Knight	Ixalan	3	1	1		Flash[LINE BREAK]Bloodcrazed Paladin enters the battlefield with a +1/+1 counter on it for each creature that died this turn.
Blossom Dryad	[2][Green]	CF	3	Creature	Dryad	Ixalan	1	2	2		[Tap]: Untap target land.
Bonded Horncrest	[3][Red]	CM	4	Creature	Dinosaur	Ixalan	2	5	5		Bonded Horncrest can't attack or block alone.
Boneyard Parley	[5][Black][Black]	CS	7	Sorcery	None	Ixalan	4				Exile up to five target creature cards from graveyards. An opponent separates those cards into two piles. Put all cards from the pile of your choice onto the battlefield under your control and the rest into their owners' graveyards.
Brazen Buccaneers	[3][Red]	CM	4	Creature	Human Pirate	Ixalan	1	2	2		Haste[LINE BREAK]When Brazen Buccaneers enters the battlefield| it explores. (Reveal the top card of your library. Put that card into your hand if it's a land. Otherwise| put a +1/+1 counter on this creature| then put the card back or put it into your graveyard.)
Bright Reprisal	[4][White]	CP	5	Instant	None	Ixalan	2				Destroy target attacking creature.[LINE BREAK]Draw a card.
Burning Sun's Avatar	[3][Red][Red][Red]	CM	6	Creature	Dinosaur Avatar	Ixalan	3	6	6		When Burning Sun's Avatar enters the battlefield| it deals 3 damage to target opponent or planeswalker and 3 damage to up to one
Call to the Feast	[2][White][Black]	CPS	4	Sorcery	None	Ixalan	2				Create three 1/1 white Vampire creature tokens with lifelink.
Cancel	[1][Blue][Blue]	CI	3	Instant	None	Ixalan	1				Counter target spell.
Captain Lannery Storm	[2][Red]	CM	3	Legendary Creature	Human Pirate	Ixalan	3	2	2		Haste[LINE BREAK]Whenever Captain Lannery Storm attacks| create a colorless Treasure artifact token with "[Tap]| Sacrifice this artifact: Add one mana of any color."[LINE BREAK]Whenever you sacrifice a Treasure| Captain Lannery Storm gets +1/+0 until end of turn.
Captivating Crew	[3][Red]	CM	4	Creature	Human Pirate	Ixalan	3	4	3		[3][Red]: Gain control of target creature an opponent controls until end of turn. Untap that creature. It gains haste until end of turn. Activate this ability only any time you could cast a sorcery.
Carnage Tyrant	[4][Green][Green]	CF	6	Creature	Dinosaur	Ixalan	4	7	6		This spell can't be countered.[LINE BREAK]Trample| hexproof
Castaway's Despair	[3][Blue]	CI	4	Enchantment	Aura	Ixalan	1				Enchant creature[LINE BREAK]When Castaway's Despair enters the battlefield| tap enchanted creature.[LINE BREAK]Enchanted creature doesn't untap during its controller's untap step.
Charging Monstrosaur	[4][Red]	CM	5	Creature	Dinosaur	Ixalan	2	5	5		Trample| haste
Chart a Course	[1][Blue]	CI	2	Sorcery	None	Ixalan	2				Draw two cards. Then discard a card unless you attacked with a creature this turn.

Figure 1: Screenshot of "Condensed_Cards_Database.csv"

In [3]:
```
# Condensing_Cards_Database.py
#
# Created: 03/21/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# This program condenses the cards database by deleting rows with the same nam
e, except the first row.
# I manually assigned mana types for lands.
#
# Inputs: Cards_Database.csv
# Dependencies: pandas
# Outputs: Condensed_Cards_Database.csv


# Allow use of the pandas.read_csv method.
import pandas as pd


# Load cards database from file into a pandas DataFrame.
cards_database = pd.read_csv("Cards_Database.csv", header=0, index_col=0)

# Drop duplicate rows.
condensed_cards_database = cards_database[~cards_database.index.duplicated(kee
p="first")]

# Write the condensed cards database to file.
condensed_cards_database.to_csv("Condensed_Cards_Database.csv")
```

2.2. Creating a Raw Table of Deck URL's and Any Win / Loss Ratios

Return to Tasks Checklist

2.2.1. Task, Software, and Result

I wrote the below Python program to create a raw table of URL's to information on 4,827 proven or unproven decks read from https://mtgarena.pro/decks/?community/ (https://mtgarena.pro/decks/?community/) and any win / loss ratios corresponding to those decks. A cleaned table will be created from this raw table. Please see the below image for what my raw table of URL's and win / loss ratios looks like.

My program requires no inputs. My program depends on using the Selenium toolset to automate Google Chrome in scrolling through a webpage of dynamically loaded deck information living at http://mtgarena.pro/decks/?community/ (http://mtgarena.pro/decks/?community/). My program depends on using the time Python module to put itself to sleep for six seconds after each scroll to give the webpage sufficient time to load new data and, actually, all previous data as well. My program depends on using the BeautifulSoup Python library to structure HTML representing the webpage. My program relies on the re Python module to find deck ID numbers. My program outputs my raw table of deck URL's and any win / loss ratios to "URLs_n_Ratios.csv" in the "Data_With_Ravnica_Allegiance" subfolder to this notebook's folder.

Deck-Page URL	Win / Loss Ratio
https://mtgarena.pro/decks/esper-anti-grave-10	??
https://mtgarena.pro/decks/naya-turbolands	??
https://mtgarena.pro/decks/goblin-storm-2	??
https://mtgarena.pro/decks/mono-black-aggro-38	??
https://mtgarena.pro/decks/dinos-aggro	??
https://mtgarena.pro/decks/d5b65e0b27c6464-1	??
https://mtgarena.pro/decks/jund-fun	52.8
https://mtgarena.pro/decks/gr-aggro-by-gerry-thompson	??
https://mtgarena.pro/decks/gu-disruptive-aggro-by-mtgnerdgirl	??
https://mtgarena.pro/decks/gates-387	??
https://mtgarena.pro/decks/kaya-20	??
https://mtgarena.pro/decks/simic-midrange-breed	??
https://mtgarena.pro/decks/mono-red-aggro-273	55.4
https://mtgarena.pro/decks/mono-u-50	54
https://mtgarena.pro/decks/green-stomp-1	??
https://mtgarena.pro/decks/simic-counters-39	0
https://mtgarena.pro/decks/esper-control-685	??
https://mtgarena.pro/decks/black-singleton-liiana	??
https://mtgarena.pro/decks/gruul-bo1-21	??
https://mtgarena.pro/decks/mono-red-149	56.2
https://mtgarena.pro/decks/lifegain-acuity	??
https://mtgarena.pro/decks/selesnya-tokens-338	??
https://mtgarena.pro/decks/token-27	??
https://mtgarena.pro/decks/tri-explore	??
https://mtgarena.pro/decks/bu-surveil-12	??
https://mtgarena.pro/decks/mono-blue-aggro-82	51.6
https://mtgarena.pro/decks/abzan-token-2	??
https://mtgarena.pro/decks/esper-control-673	??

Figure 2: Screenshot of "URLs_n_Ratios.csv"

In [7]:
```
# Creating_a_Raw_Table_of_Deck_URLs_and_Any_Win-Loss_Ratios.py
#
# Created: 01/07/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# This program records the dynamically loaded HTML of MTGArena.pro/decks/?comm
unity/,
# a webpage presenting introductory information for at least 4,827 MTG decks.
# The program then creates a table of deck URL's and win / loss ratios.
#
# Inputs: None
# Dependencies: selenium.webdriver, time, bs4.BeautifulSoup, re
# Outputs: URLs_n_Ratios.csv,
# which contains a 4,828 x 2 table. One row is the table header.


# Allow creation of webdriver class instances.
from selenium import webdriver

# Allow use of the time.sleep method.
import time

# Allow creation of BeautifulSoup class instances.
from bs4 import BeautifulSoup

# Allow use of the re.compile method.
import re


#############################################################################
# Extract the dynamically loaded HTML of MTGArena.pro/decks/?community/.
#############################################################################

# Open a Chrome browser.
browser = webdriver.Chrome()

# Navigate to the webpage.
browser.get("https://mtgarena.pro/decks/?community")

# Initialize last_height as the webpage's present scroll height.
last_height = browser.execute_script("return document.body.scrollHeight")

# Set a time for the program to sleep between scrolls to allow for the webpage
to load.
sleep_time  = 6

while True:

    # Scroll down to bottom of the body.
    browser.execute_script("window.scrollTo(0, document.body.scrollHeight);")

    # Wait to load page.
    time.sleep(sleep_time)

    # Calculate new scroll height.
    new_height = browser.execute_script("return document.body.scrollHeight")
```

```
    # If new scroll height equals last scroll height,
    # then the webpage hasn't loaded any more information,
    # and it's time to end the scrolling.
    if new_height == last_height:
        break

    # If the webpage loaded information, then set the last height to the new h
eight.
    last_height = new_height

# Extract the HTML representing the fully loaded webpage.
inner_HTML = browser.execute_script("return document.body.innerHTML")

# Close the browser.
browser.close()


################################################################
# Write each deck URL and win / loss ratio in the loaded HTML
# into a table in a CSV file.
################################################################

# Parse the loaded HTML into a BeautifulSoup.
soup = BeautifulSoup(inner_HTML, "html.parser")

# Open a CSV file for writing.
f = open("./Data_With_Ravnica_Allegiance/URLs_n_Ratios.csv", "w", encoding="ut
f-8")

# Write table headers to the CSV file.
f.write("Deck-Page URL" + "," + "Win / Loss Ratio" + "\n")

# For each division block in the BeautifulSoup
# that has an ID containing "deckxx" and that
# represents a deck...
for div in soup.findAll('div', id=re.compile('deckxx')):

    # Find the URL associated with that deck.
    deck_url = "https://mtgarena.pro" + div.find('a')['href']

    # Find the win ratio associated with that deck.
    win_ratio = (div.contents)[5].text
    if win_ratio != "??":
        win_ratio = float(win_ratio[:-1])

    # Write the deck URL and deck win ratio to the CSV file.
    f.write(deck_url + "," + str(win_ratio) + "\n")

# Close the file.
f.close()
```

2.2.2. Ensuring the Best Result

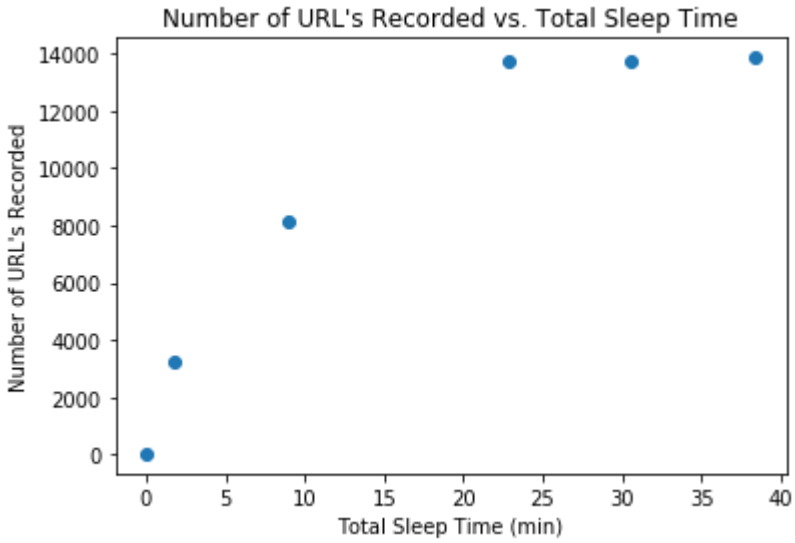
This section applies to data loaded by the above program when https://mtgarena.pro/decks/?community/ only listed decks containing cards from the Ixalan, Rivals of Ixalan, Dominaria, Core Set 2019, and Guilds of Ravnica card sets. I believe that sometime around 03/01/19, a month or so after the Ravnica Allegiance card set was added to MTG Arena, MTGArena.pro reset its community decks listing.

I wrote the below Python program to ensure that I would not gain significantly more URL's and win ratios than 13,854 if I changed the sleep time between scrolls through the http://mtgarena.pro/decks/?community/ (http://mtgarena.pro/decks/?community/) webpage from 5 seconds to 6 seconds. My program generates a graph showing a logarithmic relationship between number of URL's recorded (in HTML) and total sleep time (the product of sleep time per scroll and the number of sleeps before the program above didn't sleep long enough). You can see below that there is a relatively minute difference in numbers of decks recorded for total sleep times of 24 minutes, 31 minutes, and 39 minutes.

My program requires information on the number of URL's recorded in HTML for each sleep time per scroll. My program depends on using the numpy Python package to organize this information. My program depends on using the matplotlib.pyplot Python interface to graph "Number of URL's Recorded vs. Total Sleep Time". My program ouputs this graph in this notebook.

In [1]:
```
# Graph_Num_URLs_Recorded_vs_Total_Sleep_Time.py
#
# Created: 01/??/19 by Tom Lever
# Updated: 03/19/19 by Tom Lever
#
# This program graphs the number of URL's recorded in HTML versus total sleep
 time.
# This graph is useful in determining that having a sleep time per load of 5 s
econds
# is about the maximum sleep time per load that I want, given that I can extra
polate
# that there won't be too much more gain in URL's for larger sleep times per s
croll.
#
# Inputs: Array of numbers of URL's recorded for six runs of the program abov
e,
# each with a unique sleep time per scroll.
# Dependencies: numpy, matplotlib.pyplot
# Outputs: Graph of Number of Decks Recorded vs. Total Sleep Time


# Allow creation of ndarrays.
import numpy as np

# Allow use of matplotlib.pyplot scatter-plotting and labeling methods.
import matplotlib.pyplot as plt


#########################################################################
# Assemble arrays for number of decks loaded vs. sleep times to load.
#########################################################################

# Create an ndarray of the number of URL's recorded for each run.
nums_URLs_recorded = np.array([30, 3240, 8130, 13737, 13763, 13854])

# Calculate the total sleep time corresponding to each number of URL's recorde
d.
total_sleep_times = np.zeros(nums_URLs_recorded.shape[0])
decks_added_per_scroll = 30
seconds_per_minute = 60
for i in range(0, nums_URLs_recorded.shape[0]):
    total_sleep_times[i] = i*(nums_URLs_recorded[i]/decks_added_per_scroll - 1
)*(1/seconds_per_minute)


#########################################################################################
# Graph Number of Decks Loaded vs. Minutes of Sleeping Before Loading Failure.
#########################################################################################

# This magic function is apparently important
# in getting the scatter plot to show in Jupyter Notebook.
# The ordering of the following three code blocks is important.
%matplotlib inline

plt.scatter(total_sleep_times, nums_URLs_recorded)
plt.xlabel("Total Sleep Time (min)")
```

```
plt.ylabel("Number of URL's Recorded")
plt.title("Number of URL's Recorded vs. Total Sleep Time")

# This function is important in getting the scatter plot
# to show without scatter-plot object information.
plt.show()
```



2.3. Creating a Cleaned Table of Deck URL's and Win / Loss Ratios

Return to Tasks Checklist

I wrote the below Python program to create a cleaned table of URL's to information on 2,412 proven decks read from https://mtgarena.pro/decks/?community/ (https://mtgarena.pro/decks/?community/) and the win / loss ratios corresponding to those decks. The URL's will be used first to archive information on each deck to its own text file. The win / loss ratios will be used first in a table of deck URL's and win / loss ratios corresponding to decks ensured to have sixty cards. Please see the below screenshot of the cleaned table of URL's and ratios.

My program requires the path of my generated raw table of URL's and win / loss ratios. My program depends on using the pandas Python library to load the raw table and to eliminate all rows with the value of "??", instances of which I translate into "NaN". My program depends on using the numpy Python package to replace all instances of "??" with "NaN". My program outputs my cleaned table of deck URL's and win / loss ratios to "URLs_n_Ratios--Cleaned.csv" in the "Data_With_Ravnica_Allegiance" subfolder to this notebook's folder.

Deck-Page URL	Win / Loss Ratio
https://mtgarena.pro/decks/jund-fun	52.8
https://mtgarena.pro/decks/mono-red-aggro-273	55.4
https://mtgarena.pro/decks/mono-u-50	54
https://mtgarena.pro/decks/simic-counters-39	0
https://mtgarena.pro/decks/mono-red-149	56.2
https://mtgarena.pro/decks/mono-blue-aggro-82	51.6
https://mtgarena.pro/decks/for-the-gods	0
https://mtgarena.pro/decks/orzhov-anti-aggro-by-magicshibby	63.6
https://mtgarena.pro/decks/mardu-hero-of-precinct-one-by-magicshibby	49.3
https://mtgarena.pro/decks/grixis-discard-78	30.1
https://mtgarena.pro/decks/mono-black-like-dev-did-it	29.9
https://mtgarena.pro/decks/vol-be2e042e-f4d8-455d-a364-4445c466b59e	54.5
https://mtgarena.pro/decks/dredge-41	41.7
https://mtgarena.pro/decks/esper-aquity-1	0
https://mtgarena.pro/decks/sultai-midrange-560	56
https://mtgarena.pro/decks/esper-historic-14	42.6
https://mtgarena.pro/decks/mono-u-43	51.2
https://mtgarena.pro/decks/blink-orz	66.7
https://mtgarena.pro/decks/mno-blue	52.2
https://mtgarena.pro/decks/bg-bo3-3	60
https://mtgarena.pro/decks/mythic-boros-angels	71.4
https://mtgarena.pro/decks/white-aggggrri	57.7
https://mtgarena.pro/decks/-672	51.4
https://mtgarena.pro/decks/dragon-trump-by-megamogwai	31
https://mtgarena.pro/decks/tesper-control	51.5
https://mtgarena.pro/decks/rakdos-ii-1	50
https://mtgarena.pro/decks/esper-tempo-27	40.3
https://mtgarena.pro/decks/gateshift-16	45.2

Figure 3: Screenshot of "URLs_n_Ratios--Cleaned.csv"

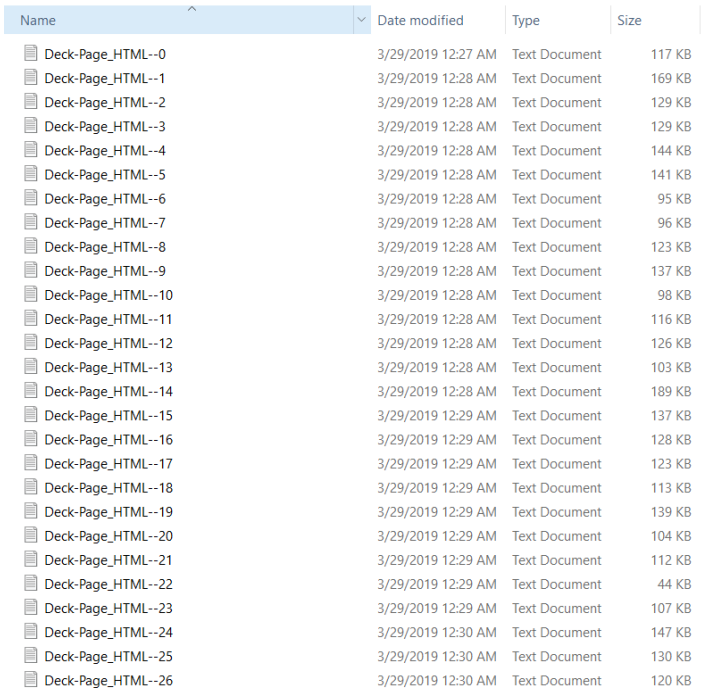
In [5]:
```
# Creating_a_Cleaned_Table_of_Deck_URLs_and_Win-Loss_Ratios.py
#
# Created: 01/09/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# This program generates a version of "URLs_n_Ratios.csv" with rows with no wi
n ratios removed.
#
# Inputs: "URLs_n_Ratios.csv"
# Dependencies: pandas, numpy
# Outputs: "URLs_n_Ratios--Cleaned.csv",
# which contains a 2,413 x 2 table. One row is the table header.


# Allows using the read_csv, dropna, and to_csv methods.
import pandas as pd

# Allows replacing "??" with NaN.
import numpy as np


##############################################################################
##
# Read and manipulate the raw table of deck URL's and win ratios as a datafram
e.
##############################################################################
##

# Reads the raw table of deck URL's and win ratios into a dataframe.
deck_URLs_and_win_ratios = pd.read_csv("./Data_With_Ravnica_Allegiance/URLs_n_
Ratios.csv", header=0, index_col=0)

# Replaces all win ratios of "??" with NaN.
table_with_quest_marks_replaced = deck_URLs_and_win_ratios.replace("??", np.na
n)

# Removes rows with values of NaN.
table_with_NaNs_dropped = table_with_quest_marks_replaced.dropna()


############################################################
# Write the cleaned deck-page URL's and win ratios to file.
############################################################

table_with_NaNs_dropped.to_csv("./Data_With_Ravnica_Allegiance/URLs_n_Ratios--
Cleaned.csv")
```

2.4. Archiving Information on Each Proven Deck

Return to Tasks Checklist

I wrote the below Python program to load the webpage for each of the 2,412 proven decks and to download the HTML representing the loaded webpage into a text file corresponding to that deck. A set of 2,271 text files will be created from the set of 2,412 text files. Each text file in the set of 2,271 is ensured to correspond to a sixty card deck and to contain no sideboard information. Please see below screenshot of the folder of deck-page HTML's for each of the 2,412 proven decks.

My program requires the path of my cleaned table of URL's and win / loss ratios. My program depends on using the pandas Python library to load the cleaned table. My program depends on using the Selenium toolset to automate Google Chrome in loading webpages of deck information. My program outputs a folder of "Deck-Page_HTMLs" in the "Data_Pre_Ravnica_Allegiance" subfolder of this notebook's folder.



Name	Date modified	Type	Size
Deck-Page_HTML--0	3/29/2019 12:27 AM	Text Document	117 KB
Deck-Page_HTML--1	3/29/2019 12:28 AM	Text Document	169 KB
Deck-Page_HTML--2	3/29/2019 12:28 AM	Text Document	129 KB
Deck-Page_HTML--3	3/29/2019 12:28 AM	Text Document	129 KB
Deck-Page_HTML--4	3/29/2019 12:28 AM	Text Document	144 KB
Deck-Page_HTML--5	3/29/2019 12:28 AM	Text Document	141 KB
Deck-Page_HTML--6	3/29/2019 12:28 AM	Text Document	95 KB
Deck-Page_HTML--7	3/29/2019 12:28 AM	Text Document	96 KB
Deck-Page_HTML--8	3/29/2019 12:28 AM	Text Document	123 KB
Deck-Page_HTML--9	3/29/2019 12:28 AM	Text Document	137 KB
Deck-Page_HTML--10	3/29/2019 12:28 AM	Text Document	98 KB
Deck-Page_HTML--11	3/29/2019 12:28 AM	Text Document	116 KB
Deck-Page_HTML--12	3/29/2019 12:28 AM	Text Document	126 KB
Deck-Page_HTML--13	3/29/2019 12:28 AM	Text Document	103 KB
Deck-Page_HTML--14	3/29/2019 12:28 AM	Text Document	189 KB
Deck-Page_HTML--15	3/29/2019 12:29 AM	Text Document	137 KB
Deck-Page_HTML--16	3/29/2019 12:29 AM	Text Document	128 KB
Deck-Page_HTML--17	3/29/2019 12:29 AM	Text Document	123 KB
Deck-Page_HTML--18	3/29/2019 12:29 AM	Text Document	113 KB
Deck-Page_HTML--19	3/29/2019 12:29 AM	Text Document	139 KB
Deck-Page_HTML--20	3/29/2019 12:29 AM	Text Document	104 KB
Deck-Page_HTML--21	3/29/2019 12:29 AM	Text Document	112 KB
Deck-Page_HTML--22	3/29/2019 12:29 AM	Text Document	44 KB
Deck-Page_HTML--23	3/29/2019 12:29 AM	Text Document	107 KB
Deck-Page_HTML--24	3/29/2019 12:30 AM	Text Document	147 KB
Deck-Page_HTML--25	3/29/2019 12:30 AM	Text Document	130 KB
Deck-Page_HTML--26	3/29/2019 12:30 AM	Text Document	120 KB

</td>

Figure 4: Screenshot of "./Data_With_Ravnica_Allegiance/Deck-Page_HTMLs"

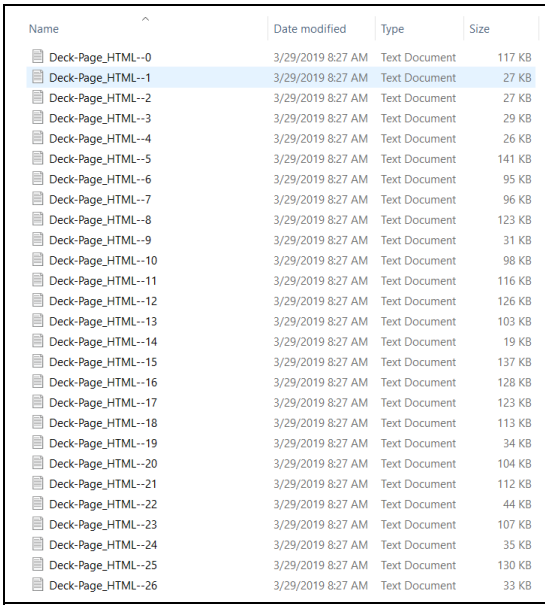
In [15]:
```
# Archiving_Information_on_Each_Proven_Deck.py
#
# Created: 01/??/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# This program reads the series of URL's from "URLs_n_Ratios--Cleaned.csv"
# and archives into a common folder text files containing the HTMLs
# representing the web pages accessed via each URL.
#
# Inputs: URLs_n_Ratios--Cleaned.csv
# Dependencies: pandas, selenium.webdriver
# Outputs: 2,412 text files containing HTMLs


# Allow use of the pd.read_csv method.
import pandas as pd

# Allow creation of a webdriver class instance.
from selenium import webdriver


# Read the URLs from "URLs_n_Ratios--Cleaned.csv" into a pandas series.
urls = pd.read_csv("./Data_With_Ravnica_Allegiance/URLs_n_Ratios--Cleaned.csv"
)["Deck-Page URL"]

# Open a Chrome browser.
browser = webdriver.Chrome()

# For each MTG deck...
for i in range(0, len(urls)):

    # Open a text file to store the HTML representing the web page for that de
ck.
    f = open("./Data_With_Ravnica_Allegiance/Deck-Page_HTMLs/Deck-Page_HTML--"
+ str(i) + ".txt", "w", encoding="utf-8")

    # Navigate to a webpage via the URL.
    browser.get(urls[i])

    # Extract the loaded HTML of the web page and add it to the file.
    f.write(browser.execute_script("return document.body.innerHTML"))

    # Close the file.
    f.close()

# Close the browser.
browser.close()
```

2.5. Archiving Information on Sixty-Card Decks and Winnowing Cleaned Table of URLs and Ratios

Return to Tasks Checklist

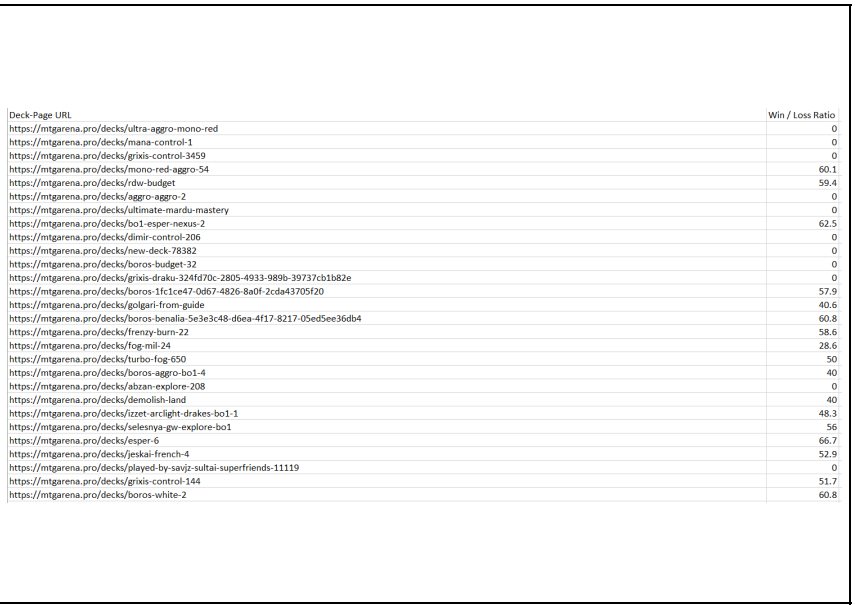
I wrote the below Python program to archive information on each deck ensured to have sixty cards, and to create a version of "URLs_n_Ratios--Cleaned.csv" that lists URLs to and win ratios for decks ensured to have sixty cards. The program ensures each deck has exactly sixty cards by finding a "60 / 60" string in the deck's HTML. The program additionally ensures that it does not confuse sideboard cards with deck cards by eliminating, for each deck, all text mentioning sideboard and all text thereafter. The archive will be used first to create a list of ID numbers and names for cards in the 2,271 decks. The winnowed table of URLs and ratios will be used first as the multirow header of the (card ID / card number) $\times$ (deck URL / win/loss ratio) database. Please see below screenshots of the folder of deck-page HTML's for each of the 2,271 decks ensured to have sixty cards, and of "Winnowed_URLs_n_Ratios.csv".

My program requires the path of my cleaned table of URL's and win / loss ratios. My program depends on using the pandas Python library to load the cleaned table. My program depends on using the re Python module to find all text before sideboard-related text. My program outputs a folder of deck-page HTMLs corresponding to sixty-card decks and "Winnowed_URLs_n_Ratios.csv" in the "Data_With_Ravnica_Allegiance" subfolder of this notebook's folder.



Name	Date modified	Type	Size
Deck-Page_HTML--0	3/29/2019 8:27 AM	Text Document	117 KB
Deck-Page_HTML--1	3/29/2019 8:27 AM	Text Document	27 KB
Deck-Page_HTML--2	3/29/2019 8:27 AM	Text Document	27 KB
Deck-Page_HTML--3	3/29/2019 8:27 AM	Text Document	29 KB
Deck-Page_HTML--4	3/29/2019 8:27 AM	Text Document	26 KB
Deck-Page_HTML--5	3/29/2019 8:27 AM	Text Document	141 KB
Deck-Page_HTML--6	3/29/2019 8:27 AM	Text Document	95 KB
Deck-Page_HTML--7	3/29/2019 8:27 AM	Text Document	96 KB
Deck-Page_HTML--8	3/29/2019 8:27 AM	Text Document	123 KB
Deck-Page_HTML--9	3/29/2019 8:27 AM	Text Document	31 KB
Deck-Page_HTML--10	3/29/2019 8:27 AM	Text Document	98 KB
Deck-Page_HTML--11	3/29/2019 8:27 AM	Text Document	116 KB
Deck-Page_HTML--12	3/29/2019 8:27 AM	Text Document	126 KB
Deck-Page_HTML--13	3/29/2019 8:27 AM	Text Document	103 KB
Deck-Page_HTML--14	3/29/2019 8:27 AM	Text Document	19 KB
Deck-Page_HTML--15	3/29/2019 8:27 AM	Text Document	137 KB
Deck-Page_HTML--16	3/29/2019 8:27 AM	Text Document	128 KB
Deck-Page_HTML--17	3/29/2019 8:27 AM	Text Document	123 KB
Deck-Page_HTML--18	3/29/2019 8:27 AM	Text Document	113 KB
Deck-Page_HTML--19	3/29/2019 8:27 AM	Text Document	34 KB
Deck-Page_HTML--20	3/29/2019 8:27 AM	Text Document	104 KB
Deck-Page_HTML--21	3/29/2019 8:27 AM	Text Document	112 KB
Deck-Page_HTML--22	3/29/2019 8:27 AM	Text Document	44 KB
Deck-Page_HTML--23	3/29/2019 8:27 AM	Text Document	107 KB
Deck-Page_HTML--24	3/29/2019 8:27 AM	Text Document	35 KB
Deck-Page_HTML--25	3/29/2019 8:27 AM	Text Document	130 KB
Deck-Page_HTML--26	3/29/2019 8:27 AM	Text Document	33 KB

Figure 5.1: Screenshot of "./Data_With_Ravnica_Allegiance/HTMLs--60--Before_Sideboards"



Deck-Page URL	Win / Loss Ratio
https://mtgarena.pro/decks/ultra-aggro-mono-red	0
https://mtgarena.pro/decks/mana-control-1	0
https://mtgarena.pro/decks/grixis-control-3459	0
https://mtgarena.pro/decks/mono-red-aggro-54	60.1
https://mtgarena.pro/decks/rdw-budget	59.4
https://mtgarena.pro/decks/aggro-aggro-2	0
https://mtgarena.pro/decks/ultimate-mardu-mastery	0
https://mtgarena.pro/decks/bo1-esper-nexus-2	62.5
https://mtgarena.pro/decks/dimir-control-206	0
https://mtgarena.pro/decks/new-deck-78382	0
https://mtgarena.pro/decks/boros-budget-32	0
https://mtgarena.pro/decks/grixis-draku-324fd70c-2805-4933-989b-39737cb1b82e	0
https://mtgarena.pro/decks/boros-1fc1ce47-0d67-4826-8a0f-2cda43705f20	57.9
https://mtgarena.pro/decks/golgari-from-guide	40.6
https://mtgarena.pro/decks/boros-benalia-5e3e3c48-d6ea-4f17-8217-05ed5ee36db4	60.8
https://mtgarena.pro/decks/frenzy-burn-22	58.6
https://mtgarena.pro/decks/fog-mil-24	28.6
https://mtgarena.pro/decks/turbo-fog-650	50
https://mtgarena.pro/decks/boros-aggro-bo1-4	40
https://mtgarena.pro/decks/abzan-explore-208	0
https://mtgarena.pro/decks/demolish-land	40
https://mtgarena.pro/decks/izzet-arclight-drakes-bo1-1	48.3
https://mtgarena.pro/decks/selesnya-gw-explore-bo1	56
https://mtgarena.pro/decks/esper-6	66.7
https://mtgarena.pro/decks/jeskai-french-4	52.9
https://mtgarena.pro/decks/played-by-savjz-sultai-superfriends-11119	0
https://mtgarena.pro/decks/grixis-control-144	51.7
https://mtgarena.pro/decks/boros-white-2	60.8

Figure 5.2: Screenshot of "Winnowed_URLs_and_Ratios.csv"

In [ ]:
```
# Archiving_Information_on_Sixty-Card_Decks_and_Winnowing_URL_and_Win-Loss_Rat
io_Table.py
#
# Created: 01/16/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# This program reads each Deck-Page_HTML-- file and
# creates a version of that file, if that file has a 60-card deck,
# with HTML before the sideboard, if a sideboard exists,
# or all the HTML, if a sideboard does not exist.
#
# This program also winnows "URLs_n_Ratios--Cleaned.csv" to "Winnowed_URLs_n_R
atios.csv",
# which contains a table of URLs and ratios corresponding to decks ensured to
 have sixty cards.
#
# Inputs: Deck-Page_HTML-- files, URLs_n_Ratios--Cleaned.csv
# Dependencies: pandas, re
# Outputs: 2,271 text files containing winnowed deck-page HTML's, and "Winnowe
d_URLs_n_Ratios.csv".
# Winnowed_URLs_n_Ratios contains a 5,391 x 2 table. One row is the table head
er.


# Allows use of the pandas.read_csv method.
import pandas as pd

# Allows use of the re.search method.
import re


# Read the cleaned URL / win ratio table into a dataframe.
deck_urls_and_win_ratios = pd.read_csv("./Data_Pre_Ravnica_Allegiance/URLs_n_R
atios--Cleaned.csv")

# Enter the "Deck-Page URL" column into a list.
deck_urls = deck_urls_and_win_ratios["Deck-Page URL"].tolist()

# Enter the "Win / Loss Ratio" column into a list.
win_ratios = deck_urls_and_win_ratios["Win / Loss Ratio"].tolist()

# Create an empty list for recording the deck-page URL's for sixty-card decks.
winnowed_deck_urls = []

# Create an empty list for recording the win ratios for sixty-card decks.
winnowed_win_ratios = []


# Start a counter for appending to the end of each Deck-Page_HTML-- file corre
sponding to a sixty-card deck.
count = 0

# For each original Deck-Page_HTML-- file...
for i in range(0, 2411):

    # Read that original Deck-Page_HTML-- file in.
```

```
    f = open("./Data_With_Ravnica_Allegiance/Deck-Page_HTMLs/Deck-Page_HTML--"
+ str(i) + ".txt", "r", encoding="utf-8")
    deck_HTML = f.read()
    f.close()

    # If the present original Deck-Page_HTML-- file represents a 60-card dec
k...
    if "60 / 60" in deck_HTML:

        # Write the represented deck's URL into the list of deck URL's for six
ty-card decks.
        winnowed_deck_urls.append(deck_urls[i])

        # Write the represented deck's win ratio into the list of win ratios f
or sixty-card decks.
        winnowed_win_ratios.append(win_ratios[i])

        # Open a new text file to contain this file exactly or a cropped versi
on,
        # depending on whether or not the represented deck has a sideboard.
        f = open("./Data_With_Ravnica_Allegiance/HTMLs--60--Before_Sideboards/
Deck-Page_HTML--" + str(count) + ".txt", "w", encoding="utf-8")

        # If the represented deck has a sideboard...
        if "dc_dhead\">Sideboard" in deck_HTML:

            # Crop the original deck file and write it into the new file.
            f.write("<div" + re.search("<div(.*)dc_dhead\">Sideboard", deck_HT
ML).group(1))

        # If the represented deck does not have a sideboard...
        else:

            # Write the original file representing a 60-card deck into the new
file.
            f.write(deck_HTML)

        # Close the new file.
        f.close()

        # Increase the suffix for winnowed deck files by 1.
        count += 1


# Write the winnowed cleaned table of URL's and win ratios into a CSV file.
f = open("/Data_With_Ravnica_Allegiance/Winnowed_URLs_n_Ratios.csv", "w", enco
ding="utf-8")
f.write("Deck-Page URL" + "," + "Win / Loss Ratio" + "\n")
for i in range(0, len(winnowed_deck_urls)):
    f.write(winnowed_deck_urls[i] + "," + str(winnowed_win_ratios[i]) + "\n")
f.close()
```

2.6. Creating a Table of Card ID Numbers and Names

Return to Tasks Checklist

I wrote the below Python program to create a table of card ID numbers and names for each card in the 2,271 decks ensured to have sixty cards. Each (ID number, name) pair will correspond to one card in at least one of the 2,271 decks ensured to have sixty cards. Each (ID number, name) pair in the table will be unique. The ID numbers and names for cards in each deck are found in the BeautifulSoup for that deck. Unfortunately, different ID numbers may have the same card name. I handle this by aggregating rows with the same card name in the (card ID / card name) $\times$ (deck URL / win/loss ratio) database. This table of ID numbers and names will be used first in the multicolumn index of the (card ID / card name) $\times$ (deck URL / win/loss ratio) database. Please see below screenshot of the ID's and names table.

My program requires the path to the folder of information on the decks ensured to have sixty cards. My program depends on the BeautifulSoup Python library to structure the HTML information for each deck. My program depends on the re Python module to find card ID numbers within specific strings. My program outputs the table of ID's and names to "IDs_and_Names.csv" in the "Data_With_Ravnica_Allegiance" subfolder of this notebook's folder.

Card ID Number	Card Name
58	Forest
86	Island
306	Mountain
324	Plains
368	Swamp
407	Academy Journeymage
409	Adeliz| the Cinder Wind
410	Adventurous Impulse
412	Amaranthine Wall
415	Arcane Flight
416	Artificer's Assistant
417	Arvad the Cursed
418	Aryel| Knight of Windgrace
419	Aven Sentry
420	Baird| Steward of Argive
421	Baloth Gorger
422	Befuddle
423	Benalish Honor Guard
424	Benalish Marshal
425	Blackblade Reforged
426	Blessed Light
428	Blink of an Eye
431	Board the Weatherlight
432	Broken Bond
435	Cabal Stronghold
436	Caligo Skin-Witch
437	Call the Cavalry
438	Cast Down

Figure 6: Screenshot of "IDs_and_Names.csv"

In [2]:
```
# Creating_a_Table_of_Card_ID_Nums_and_Names.py
#
# This program creates a list of unique card-ID-number / card-name element pai
rs across 2,271 decks.
#
# Created: 01/??/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# Inputs: Winnowed Deck-Page_HTML-- files
# Dependencies: bs4.BeautifulSoup, re
# Outputs: IDs_and_Names.csv,
# which contains a 1,351 x 2 table. One row is the header.


# Allow creation of BeautifulSoup class instances.
from bs4 import BeautifulSoup

# Allow use of the re.compile method.
import re


################################################################################
# For each of 2,271 decks,
# extract from that deck's HTML card-ID-number / card-name element pairs, and
# add those element pairs to a running list of element pairs for all decks.
################################################################################

# Initialize an empty list for card-ID-number / card-name element pairs.
list_of_element_pairs = []

# For each winnowed deck-page HTML...
for i in range(0, 2270):

    # Read that HTML into a string.
    f = open("./Data_With_Ravnica_Allegiance/HTMLs--60--Before_Sideboards/Deck
-Page_HTML--" + str(i) + ".txt", "r", encoding="utf-8")
    deck_HTML = f.read()
    f.close()

    # Parse the loaded HTML into a BeautifulSoup.
    soup = BeautifulSoup(deck_HTML, "html.parser")

    # For each division block representing a card in the HTML for the card's w
eb page...
    for div in soup.findAll('div', id=re.compile('indeckxx')):

        # Find the card's ID number.
        card_ID_num = int(div["id"].split("indeckxx")[1])

        # Find the card's title.
        card = div["title"].replace(",", "|")

        # Add to the list of card-ID-number / card-name element pairs
        # an element pair with the present card's ID number and name.
        list_of_element_pairs.append([card_ID_num, card])
```

```
######################################################################
# Create a list of unique element pairs sorted by card ID number.
# Write card numbers and card names to a CSV file.
######################################################################

# Create a list of unique element pairs.
list_of_unique_element_pairs = [list(v) for v in dict(list_of_element_pairs).i
tems()]

# Sort the list of unique element pairs by card ID number.
sorted_list_of_unique_element_pairs = sorted(list_of_unique_element_pairs, key
=lambda pair: pair[0])

# Extract unique, sorted card ID numbers from the sorted list of unique elemen
t pairs.
card_ID_nums = [col[0] for col in sorted_list_of_unique_element_pairs]

# Extract unique, sorted card names from the sorted list of unique element pai
rs.
card_names = [col[1] for col in sorted_list_of_unique_element_pairs]

# Write the card ID numbers and card names to a CSV file.
f = open("./Data_With_Ravnica_Allegiance/IDs_and_Names--2270_Decks.csv", "w",
encoding="utf-8")
f.write("Card ID Number" + "," + "Card Name" + "\n")
for i in range(0, len(card_ID_nums)):
    f.write(str(card_ID_nums[i]) + "," + card_names[i] + "\n")
f.close()
```

2.7. Creating a (Card ID / Card name) × (Deck URL / Win/Loss Ratio) Database

Return to Tasks Checklist

I wrote the below Python program to create a (card ID / card name) × (deck URL / win/loss ratio) database. This database will be used first to create a condensed database without MTGArena.pro ID numbers, without deck-page URLs, and with rows aggregated by name. Please see below screenshot of IDs_Names_x_URLs_Ratios_Database.csv.

My program requires the path to the table of deck URLs and win / loss ratios for decks ensured to have sixty cards. My program requires the path to the table of card ID numbers and card names. My program relies on the pandas Python library for reading these tables. My program requires the path to the archive of information on decks ensured to have sixty cards. My program depends on the BeautifulSoup Python library for structuring the deck information. My program relies on the re Python module to find the number of instances of each card in each deck. My program relies on the numpy Python package to store the card numbers. My program outputs the (card ID number / card name) × (deck URL / win/loss ratio) database to IDs_Names_x_URLs_Ratios_Database.csv in the "Data_With_Ravnica_Allegiance" subfolder of this notebook's folder.

		https://mtgarena.pro/decks/ultra-	https://mtgarena.pro/decks/mana	https://mtgarena.pro/decks/grixis	https://mtgarena.pro/decks/monc	https://mtgarena.pro/decks/rdw-t	https://mtgarena.pro/decks/aggrc	https://mtgarena.pro/decks/ultim	https://mtgarena.pro/decks/bo1-t	https://mtgarena.pro/decks/dimir	https://mtgarena.pro/decks/new-t	https://mtgarena.pro/decks/boros	https://mtgarena.pro/decks/grixis	https://mtgarena.pro/decks/boros	https://mtgarena.pro/decks/golga	https://mtgarena.pro/decks/boros	https://mtgarena.pro/decks/frenz	https://mtgarena.pro/decks/fog-m	https://mtgarena.pro/decks/turbo	https://mtgarena.pro/decks/boros	https://mtgarena.pro/decks/abzar	https://mtgarena.pro/decks/demo	https://mtgarena.pro/decks/izzet-	https://mtgarena.pro/decks/seles	https://mtgarena.pro/decks/esper	https://mtgarena.pro/decks/jeskai	https://mtgarena.pro/decks/playe	https://mtgarena.pro/decks/grixis	https://mtgarena.pro/decks/boros	https://mtgarena.pro/decks/golga	https://mtgarena.pro/decks/monc
		-	-	-	60.1	59.4	-	-	62.5	-	-	-	-	57.9	40.6	60.8	58.6	28.6	50.0	40.0	-	40.0	48.3	56.0	66.7	52.9	-	51.7	60.8	53.4	52.6
58	Forest	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
86	Island	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
306	Mountain	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	-	-	-
324	Plains	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
368	Swamp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
407	Academy Journeymage	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
409	Adeliz| the Cinder Wind	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
410	Adventurous Impulse	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
412	Amaranthine Wall	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
415	Arcane Flight	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
416	Artificer's Assistant	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
417	Arvad the Cursed	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
418	Aryel| Knight of Windgrace	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
419	Aven Sentry	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
420	Baird| Steward of Argive	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
421	Baloth Gorger	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
422	Befuddle	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 7: Screenshot of "IDs_Nums_x_URLs_Ratios_Database.csv"

In [7]:
```
# Creating_a_Card_ID_Card_Name_x_Deck_URL_Win-Loss_Ratio_Database.py
#
# Write to file a (card ID / card name) x (deck URL / win-loss ratio) database
# with the number of copies of each card in each deck filling in the table.
#
# Created: 01/??/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# Inputs: Winnowed_URLs_n_Ratios.csv, IDs_and_Names.csv, winnowed Deck-Page_HT
ML-- files.
# Dependencies: pandas, BeautifulSoup, re, numpy
# Outputs: IDs_Names_x_URLs_Ratios_Database.csv


# Allow use of the pd.read_csv method.
import pandas as pd

# Allow creation of a BeautifulSoup class instance.
from bs4 import BeautifulSoup

# Allow use the of the re.compile method.
import re

# Allow creation of a numpy matrix.
import numpy as np


# Import Winnowed_URLs_n_Ratios.csv.
deck_urls_and_win_ratios = pd.read_csv("./Data_With_Ravnica_Allegiance/Winnowe
d_URLs_and_Ratios.csv")

# Send the "Deck-Page URL" column to a list.
deck_urls = deck_urls_and_win_ratios["Deck-Page URL"].tolist()

# Send the "Win / Loss Ratio" column to a list.
win_ratios = deck_urls_and_win_ratios["Win / Loss Ratio"].tolist()

# Import IDs_and_Names--5930_Decks.csv.
card_ID_nums_and_names = pd.read_csv("./Data_With_Ravnica_Allegiance/IDs_and_N
ames.csv")

# Send the "Card ID Number" column to a list.
card_ID_nums = card_ID_nums_and_names["Card ID Number"].tolist()

# Send the "Card Name" column to a list.
card_names = card_ID_nums_and_names["Card Name"].tolist()

# Create a card_ID_nums x deck_urls matrix of to store the number of each card
in each deck.
matrix_of_card_nums = np.zeros((len(card_ID_nums), len(deck_urls)))


# For each winnowed deck...
for i in range(0, len(deck_urls)):

    # Create a BeautifulSoup of the HTML for the present deck.
```

```
    f = open("./Data_With_Ravnica_Allegiance/HTMLs--60--Before_Sideboards/Deck
-Page_HTML--" + str(i) + ".txt", "r", encoding="utf-8")
    deck_HTML = f.read()
    soup = BeautifulSoup(deck_HTML, "html.parser")

    # For each division block in the Soup representing a card...
    for div in soup.findAll('div', id=re.compile('indeckxx')):

        # Find that card's ID number.
        card_ID_num = int(div["id"].split("indeckxx")[1])

        # Find the number of copies of the present card in the deck.
        num_cards = int(div.find('div', {"class": "dc_ccopies dc_ccc dc_ib"}).
text)

        # Add the number of copies of each card to the appropriate cell in the
matrix of card numbers.
        matrix_of_card_nums[card_ID_nums.index(card_ID_num), i] = num_cards


# Write IDs_Names_x_URLs_Ratios_Database.csv.
f = open("./Data_With_Ravnica_Allegiance/IDs_Names_x_URLs_Ratios_Database.csv"
, "w", encoding="utf-8")

f.write(",,")
for i in range(0, len(deck_urls)-1):
    f.write(deck_urls[i] + ",")
f.write(deck_urls[len(deck_urls)-1] + "\n")

f.write(",,")
for i in range(0, len(deck_urls)-1):
    f.write(str(win_ratios[i]) + ",")
f.write(str(win_ratios[len(deck_urls)-1]) + "\n")

for i in range(0, len(card_ID_nums)):
    f.write(str(card_ID_nums[i]) + "," + card_names[i] + ",")
    for j in range(0, len(deck_urls)-1):
        f.write(str(matrix_of_card_nums[i, j]) + ",")
    f.write(str(matrix_of_card_nums[i, len(deck_urls)-1]) + "\n")

f.close()
```

2.8. Creating a Name × Ratio Database

Return to Tasks Checklist

I wrote the below Python program to condense the (card ID / card name) $\times$ (deck URL / win/loss ratio) database by eliminating the ID column, the URL row, and aggregating rows by name. This database will be used first to create a table of card names and total numbers of occurrences in 2,271 decks. Please see below screenshot of Name_x_Ratio_Database.csv.

My program requires the path to the (card ID number / card name) $\times$ (deck URL / win/loss ratio) database. My program relies on the numpy Python package for storing card numbers in a matrix. My program outputs the condensed database to Name_x_Ratio_Database.csv in the "Data_With_Ravnica_Allegiance" subfolder of this notebook's folder.

	-	-	-	60.1	59.4	-	-	62.5	-	-	-	-	57.9	40.6	60.8	58.6	28.6	50.0	40.0	-	40.0	48.3	56.0	66.7	52.9	-	51.7	60.8	53.4	52.6
Forest	-	-	-	12	-	-	-	-	-	-	-	-	7	-	4	-	-	-	-	9	-	-	-	-	-	-	-	1	-	10
Island	-	-	20	6	-	19	-	-	-	-	-	22	-	-	-	1	19	-	19	-	-	-	7	-	1	-	1	1	-	11
Mountain	2	19	-	-	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	-	-	-	-	15	-	-	24	-
Plains	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-	-	-	7	-	-	10	20	-	-	-	-	-	-	-	-
Swamp	-	-	-	-	-	-	23	8	3	2	20	-	9	1	-	1	-	6	-	5	-	-	6	-	1	-	1	-	-	-
Academy Journeymage	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Adeliz| the Cinder Wind	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Adventurous Impulse	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Amaranthine Wall	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Arcane Flight	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Artificer's Assistant	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Arvad the Cursed	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Aryel| Knight of Windgrace	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Aven Sentry	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Baird| Steward of Argive	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Baloth Gorger	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Befuddle	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Benalish Honor Guard	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Benalish Marshal	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-
Blackblade Reforged	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Blessed Light	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Blink of an Eye	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Board the Weatherlight	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Broken Bond	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Cabal Stronghold	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Caligo Skin-Witch	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Call the Cavalry	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Cast Down	-	-	-	-	-	-	-	-	2	-	-	-	-	-	3	2	-	-	-	2	-	-	3	-	3	-	1	-	-	-

Figure 8: Screenshot of "Name_x_Ratio_Database.csv"

In [43]:
```
# Condensing_ID_Name_x_URL_Ratio_Database.py
#
# This program condenses the (card ID / card name) x (deck URL / win-loss rati
o) database
# by eliminating the card ID column, eliminating the URL row, and aggregating
 rows by name.
#
# Created: 03/??/19 by Tom Lever
# Updated: 04/07/19 by Tom Lever
#
# Inputs: IDs_Names_x_URLs_Ratios_Database.csv
# Dependencies: numpy
# Outputs: Name_x_Ratio_Database.csv


# Allow creation of a numpy matrix.
import numpy as np


# Read from file the ID / name x URL / ratio database into a string.
f = open("./Data_With_Ravnica_Allegiance/IDs_Names_x_URLs_Ratios_Database.csv"
, "r", encoding="utf-8")
database = f.read()
f.close()

# Send the rows in the database into a list.
rows = list(filter(None, database.split("\n")))

# Send the row of ratios into a list.
win_ratios = list(filter(None, rows[1].split(",")))

# Send the column of names into a list.
names = []
for i in range(2, len(rows)):
    present_row = list(filter(None, rows[i].split(",")))
    names.append(present_row[1])

for i in range(0, len(names)):
    names[i] = names[i].replace("Assure // Assemble", "Assure // Assemble (Ass
emble)")
    names[i] = names[i].replace("Connive // Concoct", "Connive // Concoct (Con
coct)")
    names[i] = names[i].replace("Discovery // Dispersal", "Discovery // Disper
sal (Dispersal)")
    names[i] = names[i].replace("Expansion // Explosion", "Expansion // Explos
ion (Explosion)")
    names[i] = names[i].replace("Find // Finality", "Find // Finality (Finalit
y)")
    names[i] = names[i].replace("Flower // Flourish", "Flower // Flourish (Flo
urish)")
    names[i] = names[i].replace("Integrity // Intervention", "Integrity // Int
ervention (Intervention)")
    names[i] = names[i].replace("Invert // Invent", "Invert // Invent (Inven
t)")
    names[i] = names[i].replace("Response // Resurgence", "Response // Resurge
nce (Resurgence)")
```

```
    names[i] = names[i].replace("Status // Statue", "Status // Statue (Statu
e)")
    names[i] = names[i].replace("Bedeck // Bedazzle", "Bedeck // Bedazzle (Bed
azzle)")
    names[i] = names[i].replace("Carnival // Carnage", "Carnival // Carnage (C
arnage)")
    names[i] = names[i].replace("Collision // Colossus", "Collision // Colossu
s (Colossus)")
    names[i] = names[i].replace("Consecrate // Consume", "Consecrate // Consum
e (Consume)")
    names[i] = names[i].replace("Depose // Deploy", "Depose // Deploy (Deplo
y)")
    names[i] = names[i].replace("Incubation // Incongruity", "Incubation // In
congruity (Incongruity)")
    names[i] = names[i].replace("Repudiate // Replicate", "Repudiate // Replic
ate (Replicate)")
    names[i] = names[i].replace("Revival // Revenge", "Revival // Revenge (Rev
enge)")
    names[i] = names[i].replace("Thrash // Threat", "Thrash // Threat (Threa
t)")
    names[i] = names[i].replace("Warrant // Warden", "Warrant // Warden (Warde
n)")

# Send the card-number information that forms the body of the database into a
 matrix.
matrix_of_card_nums = np.zeros((len(names), len(win_ratios)))
for i in range(2, len(rows)):
    present_row = list(filter(None, rows[i].split(",")))
    for j in range(2, len(present_row)):
        matrix_of_card_nums[i-2, j-2] = present_row[j]

# Aggregate names and matrix of card numbers by name.
# Card numbers aggregated with other card numbers are added together.
names_aggregated = names
matrix_as_list = matrix_of_card_nums.tolist()
i = 0
while i != len(names_aggregated):
    for j in range(0, i):
        if names_aggregated[i] == names_aggregated[j]:
            for k in range(0, len(matrix_as_list[1])):
                matrix_as_list[j][k] += matrix_as_list[i][k]
            del names_aggregated[i]
            del matrix_as_list[i]
            i -= 1
            break
    i += 1

# Write to file the database with rows aggregated by name.
f = open("./Data_With_Ravnica_Allegiance/Name_x_Ratio_Database.csv", "w", enco
ding="utf-8")

f.write(",")
for i in range(0, len(win_ratios)-1):
    f.write(str(win_ratios[i]) + ",")
f.write(str(win_ratios[len(win_ratios)-1]) + "\n")

for i in range(0, len(names_aggregated)):
```

```
    f.write(names_aggregated[i] + ",")
    for j in range(0, len(win_ratios)-1):
        f.write(str(matrix_as_list[i][j]) + ",")
    f.write(str(matrix_as_list[i][len(win_ratios)-1]) + "\n")

f.close()
```

2.9. Creating a Table of Card Names, Total Occurrences, and Frequencies

Return to Tasks Checklist

I wrote the below Python program to create a table of card names, total numbers of occurrences, and average frequencies in 2,271 decks. This database will be used first to create my list of best cards. Please see below screenshot of Names_Total_Occurrences_and_Freqs.csv.

My program requires the path to the name $\times$ ratio database. My program relies on the numpy Python package for storing card numbers in a matrix. My program outputs the table to Names_Total_Occurrences_and_Freqs.csv in the "Data_With_Ravnica_Allegiance" subfolder of this notebook's folder.

Name	Total Number of Occurrences	Average Frequency
Mountain	7722	10
Island	7057	8
Plains	5805	9
Forest	4723	7
Swamp	4589	6
Shock	1730	4
Opt	1462	4
Lightning Strike	1337	4
Watery Grave	1207	3
Breeding Pool	1186	3
Llanowar Elves	1185	4
Drowned Catacomb	1102	3
Isolated Chapel	1070	3
Godless Shrine	1048	3
Vraska's Contempt	1006	2
Hydroid Krasis	997	3
Glacial Fortress	975	3
Hallowed Fountain	973	3
Lava Coil	964	3
Steam Vents	928	3
Cast Down	916	2
Viashino Pyromancer	916	4
History of Benalia	913	4
Light Up the Stage	901	4
Ghitu Lavarunner	900	4
Sulfur Falls	886	3

Figure 9: Screenshot of "Names_Total_Occurrences_and_Freqs.csv"

In [3]:
```
# Creating_a_Table_of_Card_Names_Total_Occurrences_and_Frequencies.py
#
# This program creates a table of card names, total number of occurrences, and
average frequencies in 2,271 decks.
#
# Created: 03/??/19 by Tom Lever
# Updated: 04/05/19 by Tom Lever
#
# Inputs: Name_x_Ratio_Database.csv
# Dependencies: numpy
# Outputs: Names_Total_Occurrences_and_Freqs.csv


# Allow creation of a numpy matrix.
import numpy as np


# Read from file the condensed database into a string.
f = open("./Data_With_Ravnica_Allegiance/Name_x_URL_Ratio_Database.csv", "r",
encoding="utf-8")
condensed_database = f.read()
f.close()

# Send the rows in the condensed database into a list.
rows = list(filter(None, condensed_database.split("\n")))

# Send the row of win / loss ratios into a list.
ratios = list(filter(None, rows[0].split(",")))[1:]

# Send the column of card names into a list.
names = []
for i in range(2, len(rows)):
    present_row = list(filter(None, rows[i].split(",")))
    names.append(present_row[0])

# Send the card-number information that forms the body of the database into a
 matrix.
matrix_of_card_nums = np.zeros((len(names), len(ratios)))
for i in range(2, len(rows)):
    present_row = list(filter(None, rows[i].split(",")))
    for j in range(2, len(present_row)):
        matrix_of_card_nums[i-2, j-2] = present_row[j]

# Create an array of the total number of occurrences of each card.
total_occurrences = np.zeros(len(names))
for i in range(0, len(names)):
    for j in range(0, len(ratios)):
        if matrix_of_card_nums[i, j] != 0:
            total_occurrences[i] += matrix_of_card_nums[i, j]

# Create an array of the average frequencies of each card.
frequencies = np.zeros(matrix_of_card_nums.shape[0])
num_decks_card_in = np.zeros(matrix_of_card_nums.shape[0])
for i in range(0, matrix_of_card_nums.shape[0]):
    for j in range(0, matrix_of_card_nums.shape[1]):
        if matrix_of_card_nums[i, j] > 0:
```

```
            frequencies[i] += matrix_of_card_nums[i, j]
            num_decks_card_in[i] += 1
for i in range(0, matrix_of_card_nums.shape[0]):
    frequencies[i] /= num_decks_card_in[i]

# Write the table of card names and total numbers of occurrences to file.
f = open("./Data_With_Ravnica_Allegiance/Names_Total_Occurrences_and_Freqs.csv", "w", encoding="utf-8")
f.write("Name" + "," + "Total Number of Occurrences" + "," + "Average Frequency" + "\n")
for i in range(0, len(names)):
    f.write(names[i] + "," + str(total_occurrences[i]) + "," + str(frequencies[i]) + "\n")
f.close()
```

2.10. Creating a Best-Cards Database

Return to Tasks Checklist

I wrote the below Python program to create a best cards database. A best-cards database with an inventory column will be created using the best-cards database.

My program requires the path to the condensed cards database. My program relies on the pandas Python library for loading the condensed cards database. My program requires the path to the name, total occurrences, and frequencies database. My program outputs the best cards database to "Best_Cards.csv" in this notebook's "Data_With_Ravnica_Allegiance" subfolder.

Name	Rarity	Total Number of Occurrences	Mana Type	Average Frequency	Rules Text
Mountain	Land	7722	M	9.5096522	R
Island	Land	7057	I	7.9025756	U
Plains	Land	5805	P	8.7162162	W
Forest	Land	4723	F	6.7279202	G
Swamp	Land	4589	S	6.3122421	B
Hydroid Krasis	4	997	FI	3.4143836	When you cast this spell, you gain half X life and draw half X cards. Round down each time.[LINE BREAK]Flying[trample[LINE BREAK]Hydroid Krasis enters the battlefield with X +1/+1 counters on it.
History of Benalia	4	913	CP	3.6230159	(As this Saga enters and after your draw step[add a lore counter. Sacrifice after III.)[LINE BREAK]I II â†' Create a 2/2 white Knight creature token with vigilance.[LINE BREAK]III â†' Knights you control get +2/+1 until end of turn.
Vivien Reid	4	524	CF	2.3710407	+1: Look at the top four cards of your library. You may reveal a creature or land card from among them and put it into your hand. Put the rest on the bottom of your library in a random order.[LINE BREAK]â"3: Destroy target artifact[enchantment[or creature with flying.[LINE BREAK]â"8: You get an emblem with "Creatures you control get +2/+2 and have vigilance[trample[and indestructible."
Rekindling Phoenix	4	520	CM	3.0588235	Flying[LINE BREAK]When Rekindling Phoenix dies[create a 0/1 red Elemental creature token with "At the beginning of your upkeep[sacrifice this creature and return target card named Rekindling Phoenix from your graveyard to the battlefield. It gains haste until end of turn."
Teferi, Hero of Dominaria	4	518	CPI	2.9942197	+1: Draw a card. At the beginning of the next end step[untap up to two lands.[LINE BREAK]â"3: Put target nonland permanent into its owner's library third from the top.[LINE BREAK]â"8: You get an emblem with "Whenever you draw a card[exile target permanent an opponent controls."
Lyra Dawnbringer	4	275	CP	2.0833333	Flying[first strike[lifelink[LINE BREAK]Other Angels you control get +1/+1 and have lifelink.
Resplendent Angel	4	249	CP	2.2636364	Flying[LINE BREAK]At the beginning of each end step[if you gained 5 or more life this turn[create a 4/4 white Angel creature token with flying and vigilance.[LINE BREAK][3][White][White][White]: Until end of turn[Resplendent Angel gets +2/+2 and gains lifelink.
Carnage Tyrant	4	240	CF	1.6551724	This spell can't be countered.[LINE BREAK]Trample[hexproof
Seraph of the Scales	4	240	CPS	2.962963	Flying[LINE BREAK][White]: Seraph of the Scales gains vigilance until end of turn.[LINE BREAK][Black]: Seraph of the Scales gains deathtouch until end of turn.[LINE BREAK]Afterlife 2 (When this creature dies[create two 1/1 white and black Spirit creature tokens with flying.)
Arclight Phoenix	4	212	CM	3.7857143	Flying[haste[LINE BREAK]At the beginning of combat on your turn[if you've cast three or more instant and sorcery spells this turn[return Arclight Phoenix from your graveyard to the battlefield.
March of the Multitudes	4	211	FP	3.1029412	Convoke (Your creatures can help cast this spell. Each creature you tap while casting this spell pays for [1] or one mana of that creature's color.)[LINE BREAK]Create X 1/1 white Soldier creature tokens with lifelink.
Karn, Scion of Urza	4	205	C	1.7226891	+1: Reveal the top two cards of your library. An opponent chooses one of them. Put that card into your hand and exile the other with a silver counter on it.[LINE BREAK]â"1: Put a card you own with a silver counter on it from exile into your hand.[LINE BREAK]â"2: Create a 0/0 colorless Construct artifact creature token with "This creature gets +1/+1 for each artifact you control."
Spawn of Mayhem	4	190	CS	2.8787879	Spectacle {1}{R}(You may cast this spell for its spectacle cost rather than its mana cost if an opponent lost life this turn.)[LINE BREAK]Flying[trample[LINE BREAK]At the beginning of your upkeep[Spawn of Mayhem deals 1 damage to each player. Then if you have 10 or less life[put a +1/+1 counter on Spawn of Mayhem.
Trostani Discordant	4	179	CFP	2.4189189	Other creatures you control get +1/+1.[LINE BREAK]When Trostani Discordant enters the battlefield[create two 1/1 white Soldier creature tokens with lifelink.[LINE BREAK]At the beginning of your end step[each player gains control of all creatures they own.
Nullhide Ferox	4	148	CF	3.0204082	Hexproof[LINE BREAK]You can't cast noncreature spells.[LINE BREAK][2]: Nullhide Ferox loses all abilities until end of turn. Any player may activate this ability.[LINE BREAK]If a spell or ability an opponent controls causes you to discard Nullhide Ferox[put it onto the battlefield instead of putting it into your graveyard.
Ajani, Adversary of Tyrants	4	142	CP	1.5434783	+1: Put a +1/+1 counter on each of up to two target creatures.[LINE BREAK]â"2: Return target creature card with converted mana cost 2 or less from your graveyard to the battlefield.[LINE BREAK]â"7: You get an emblem with "At the beginning of your end step[create three 1/1 white Cat creature tokens with lifelink."
Kayal, Dizhov Usurper	4	128	CPS	1.5238095	+1: Exile up to two target cards from a single graveyard. You gain 2 life if at least one creature card was exiled this way.[LINE BREAK]â"1: Exile target nonland permanent with converted mana cost 1 or less.[LINE BREAK]â"5: Kayal, Dizhov Usurper deals damage to target player equal to the number of cards that player owns in exile and you gain that much life.
Kumena, Tyrant of Orazca	4	126	CFI	2.3333333	Tap another untapped Merfolk you control: Kumena[Tyrant of Orazca can't be blocked this turn.[LINE BREAK]Tap three untapped Merfolk you control: Draw a card.[LINE BREAK]Tap five untapped Merfolk you control: Put a +1/+1 counter on each Merfolk you control.
Biogenic Ooze	4	124	CF	2.3846154	When Biogenic Ooze enters the battlefield[create a 2/2 green Ooze creature token.[LINE BREAK]At the beginning of your end step[put a +1/+1 counter on each Ooze you control.[LINE BREAK][1][Green][Green][Green]: Create a 2/2 green Ooze creature token.
Nexus of Fate	4	121	CI	3.78125	Take an extra turn after this one.[LINE BREAK]If Nexus of Fate would be put into a graveyard from anywhere[reveal Nexus of Fate and shuffle it into its owner's library instead.
Skarrgan Hellkite	4	114	CM	2.4782609	Riot (This creature enters the battlefield with your choice of a +1/+1 counter or haste.)[LINE BREAK]Flying[LINE BREAK][3][Red]: Skarrgan Hellkite deals 2 damage divided as you choose among one or two targets. Activate this ability only if Skarrgan Hellkite has a +1/+1 counter on it.
Nicol Bolas, the Ravager	4	112	CISM	2.8717949	Flying[LINE BREAK]When Nicol Bolas, the Ravager enters the battlefield[each opponent discards a card.[LINE BREAK][4][Blue][Black][Red]: Exile Nicol Bolas, the Ravager[then return him to the battlefield transformed under his owner's control. Activate this ability only any time you could cast a sorcery.
Doom Whisperer	4	111	CS	2.0555556	Trample[LINE BREAK]Pay 2 life: Surveil 2. (Look at the top two cards of your library[then put any number of them into your graveyard and the rest on top of your library in any order.)
Aurelia, Exemplar of Justice	4	110	CMP	2.3913043	Flying[LINE BREAK]Mentor (Whenever this creature attacks[put a +1/+1 counter on target attacking creature with lesser power.)[LINE BREAK]At the beginning of combat on your turn[choose up to one target creature you control. Until end of turn[that creature gets +2/+0[gains trample[and gains vigilance if it's red.
Vraska, Golgari Queen	4	108	CSF	1.5882353	+2: You may sacrifice another permanent. If you do[you gain 1 life and draw a card.[LINE BREAK]â"3: Destroy target nonland permanent with converted mana cost 3 or less.[LINE BREAK]â"9: You get an emblem with "Whenever a creature you control deals combat damage to a player[that player loses the game."
Angrath, the Flame-Chained	4	107	CSM	1.7540984	+1: Each opponent discards a card and loses 2 life.[LINE BREAK]â"3: Gain control of target creature until end of turn. Untap it. It gains haste until end of turn. Sacrifice it at the beginning of the next end step if it has converted mana cost 3 or less.[LINE BREAK]â"8: Each opponent loses life equal to the number of cards in their graveyard.
The Immortal Sun	4	107	C	1.3084878	Players can't activate planeswalkers' loyalty abilities.[LINE BREAK]At the beginning of your draw step[draw an additional card.[LINE BREAK]Spells you cast cost [1] less to cast.[LINE BREAK]Creatures you control get +1/+1.
Rall Izzet Viceroy	4	92	CIM	1.1219512	+1: Look at the top two cards of your library. Put one of them into your hand and the other into your graveyard.[LINE BREAK]â"3: Rall Izzet Viceroy deals damage to target creature equal to the total number of instant and sorcery cards you own in exile and in your graveyard.[LINE BREAK]â"8: You get an emblem with "Whenever you cast an instant or sorcery spell[this emblem deals 4 damage to any target and you draw two cards."
Twilight Prophet	4	90	CS	1.5254237	Flying[LINE BREAK]Ascend (If you control ten or more permanents[you get the city's blessing for the rest of the game.)[LINE BREAK]At the beginning of your upkeep[if you have the city's blessing[reveal the top card of your library and put it into your hand. Each opponent loses X life and you gain X life[where X is that card's converted mana cost.
Divine Visitation	4	87	CP	2.0714286	If one or more creature tokens would be created under your control[that many 4/4 white Angel creature tokens with flying and vigilance are created instead.
Dovin, Grand Arbiter	4	84	CPI	2.1	+1: Until end of turn[whenever a creature you control deals combat damage to a player[put a loyalty counter on Dovin, Grand Arbiter.[LINE BREAK]â"1: Create a 1/1 colorless Thopter artifact creature token with flying. You gain 1 life.[LINE BREAK]â"7: Look at the top ten cards of your library. Put three of them into your hand and the rest on the bottom of your library in a random order.
Angel of Grace	4	82	CP	1.6078431	Flash[LINE BREAK]Flying[LINE BREAK]When Angel of Grace enters the battlefield[until end of turn[damage that would reduce your life total to less than 1 reduces it to 1 instead.[LINE BREAK][4][White][White]: Exile Angel of Grace from your graveyard: Your life total becomes 10.
Vraska, Relic Seeker	4	76	CSF	1.2881356	+2: Create a 2/2 black Pirate creature token with menace.[LINE BREAK]â"3: Destroy target artifact[creature[or enchantment. Create a colorless Treasure artifact token with "{Tap}: Sacrifice this artifact: Add one mana of any color."[LINE BREAK]â"10: Target player's life total becomes 1.
Liliana, Untouched by Death	4	55	CS	2.6190476	+1: Put the top three cards of your library into your graveyard. If at least one of them is a Zombie card[each opponent loses 2 life and you gain 2 life.[LINE BREAK]â"2: Target creature gets -X/-X until end of turn[where X is the number of Zombies you control.[LINE BREAK]â"3: You may cast Zombie cards from your graveyard this turn.
Domri, Chaos Bringer	4	55	CMF	1.71875	+1: Add {Red} or {Green}. If that mana is spent on a creature spell[it gains riot. (It enters the battlefield with your choice of a +1/+1 counter or haste.)[LINE BREAK]â"3: Look at the top four cards of your library. You may reveal up to two creature cards from among them and put them into your hand. Put the rest on the bottom of your library in a random order.[LINE BREAK]â"8: You get an emblem with "At the beginning of each end step[create a 4/4 red and green Beast creature token with trample."
Prime Speaker Vannifar	4	43	CFI	3.3076923	{Tap}[Sacrifice another creature: Search your library for a creature card with converted mana cost equal to 1 plus the sacrificed creature's converted mana cost[put that card onto the battlefield[then shuffle your library. Activate this ability only any time you could cast a sorcery.
Thousand-Year Storm	4	42	CIM	3	Whenever you cast an instant or sorcery spell[copy it for each other instant and sorcery spell you've cast before it this turn. You may choose new targets for the copies.

Figure 10: Screenshot of "Best_Cards.csv"

In [2]:
```
# Creating_a_Best_Cards_Database.py
#
# This program creates a best-cards database with columns for
# card name, rarity, total occurrences, mana type, average frequency, and rule
s text.
#
# Created: 03/??/19 by Tom Lever
# Updated: 04/02/19 by Tom Lever
#
# Inputs: Condensed_Cards_Database.csv, Names_Total_Occurrences_and_Freqs.csv
# Dependencies: pandas
# Outputs: Best_Cards.csv


## Allow use of the pandas.read_csv method.
import pandas as pd


# Read the condensed cards database into a dataframe.
condensed_cards_database = pd.read_csv("Condensed_Cards_Database.csv", index_c
ol="Name", header=0)

# Read the table of card names, total occurrences, and average frequencies int
o a dataframe.
names_occurrences_and_freqs = pd.read_csv("./Data_With_Ravnica_Allegiance/Name
s_Total_Occurrences_and_Freqs.csv", index_col="Name", header=0)

# Create a column-wise excerpt of the condensed cards database.
excerpt = condensed_cards_database[["Rarity", "Mana Type", "Rules Text"]]

# Merge the condensed cards database and table of names, occurrences, and freq
uencies into the best cards database.
best_cards = names_occurrences_and_freqs.merge(excerpt, left_index=True, right
_index=True)

# Reorganize the best cards database.
best_cards = best_cards[["Rarity", "Total Number of Occurrences", "Mana Type",
"Average Frequency", "Rules Text"]]

# Sort the cards in the best cards database first by rarity and second by occu
rrences.
best_cards = best_cards.sort_values(by=["Rarity", "Total Number of Occurrence
s"], ascending=False)

# Write the best cards database to a CSV file.
best_cards.to_csv("./Data_With_Ravnica_Allegiance/Best_Cards.csv")
```

2.11. Creating Best-Cards Database with Inventory

Return to Tasks Checklist

I wrote the below Excel VBA module to create a best cards database with columns for card name, rarity, total number of occurrences, mana type, average frequency, inventory, and rules text. The best-cards database will be sorted first by rarity and second by total number of occurrences in 2,271 decks. The best-cards database with inventory will be filtered into strong decks.

My VBA module requires that my best cards database and inventory database be in the same Excel Macro-Enabled Workbook. My VBA module requires the Microsoft Scripting Runtime library to be enabled.

Name	Rarity	Total Number of Occurrences	Mana Type	Average Frequency	Inventory	Rules Text
Mountain	Land	7722	M	9.509852217	0	R
Island	Land	7057	I	7.902575588	0	U
Plains	Land	5805	P	8.716216216	0	W
Forest	Land	4723	F	6.727920228	0	G
Swamp	Land	4589	S	6.312242091	0	B
Hydroid Krasis	4	997	FI	3.414383562	0	When you cast this spell| you gain half X life and draw half X cards. Round down each time.[LINE BREAK]Flying| trample[LINE BREAK]Hydroid Krasis enters the battlefield with X +1/+1 counters on it.
History of Benalia	4	913	CP	3.623015873	2	(As this Saga enters and after your draw step| add a lore counter. Sacrifice after III.)[LINE BREAK]I| II â€" Create a 2/2 white Knight creature token with vigilance.[LINE BREAK]III â€" Knights you control get +2/+1 until end of turn.
Vivien Reid	4	524	CF	2.371040724	0	+1: Look at the top four cards of your library. You may reveal a creature or land card from among them and put it into your hand. Put the rest on the bottom of your library in a random order.[LINE BREAK]â˜3: Destroy target artifact| enchantment| or creature with flying.[LINE BREAK]â˜8: You get an emblem with "Creatures you control get +2/+2 and have vigilance| trample| and indestructible."
Rekindling Phoenix	4	520	CM	3.058823529	3	Flying[LINE BREAK]When Rekindling Phoenix dies| create a 0/1 red Elemental creature token with "At the beginning of your upkeep| sacrifice this creature and return target card named Rekindling Phoenix from your graveyard to the battlefield. It gains haste until end of turn."
Teferi| Hero of Dominaria	4	518	CPI	2.994219653	0	+1: Draw a card. At the beginning of the next end step| untap up to two lands.[LINE BREAK]â˜3: Put target nonland permanent into its owner's library third from the top.[LINE BREAK]â˜8: You get an emblem with "Whenever you draw a card| exile target permanent an opponent controls."
Lyra Dawnbringer	4	275	CP	2.083333333	1	Flying| first strike| lifelink[LINE BREAK]Other Angels you control get +1/+1 and have lifelink.
Resplendent Angel	4	249	CP	2.263636364	1	Flying[LINE BREAK]At the beginning of each end step| if you gained 5 or more life this turn| create a 4/4 white Angel creature token with flying and vigilance.[LINE BREAK][3][White][White][White]: Until end of turn| Resplendent Angel gets +2/+2 and gains lifelink.
Carnage Tyrant	4	240	CF	1.655172414	1	This spell can't be countered.[LINE BREAK]Trample| hexproof
Seraph of the Scales	4	240	CPS	2.962962963	0	Flying[LINE BREAK][White]: Seraph of the Scales gains vigilance until end of turn.[LINE BREAK][Black]: Seraph of the Scales gains deathtouch until end of turn.[LINE BREAK]Afterlife 2 (When this creature dies| create two 1/1 white and black Spirit creature tokens with flying.)
Arclight Phoenix	4	212	CM	3.785714286	0	Flying| haste[LINE BREAK]At the beginning of combat on your turn| if you've cast three or more instant and sorcery spells this turn| return Arclight Phoenix from your graveyard to the battlefield.
March of the Multitudes	4	211	FP	3.102941176	1	Convoke (Your creatures can help cast this spell. Each creature you tap while casting this spell pays for [1] or one mana of that creature's color.)[LINE BREAK]Create X 1/1 white Soldier creature tokens with lifelink.
Karn| Scion of Urza	4	205	C	1.722689076	1	+1: Reveal the top two cards of your library. An opponent chooses one of them. Put that card into your hand and exile the other with a silver counter on it.[LINE BREAK]â˜1: Put a card you own with a silver counter on it from exile into your hand.[LINE BREAK]â˜2: Create a 0/0 colorless Construct artifact creature token with "This creature gets +1/+1 for each artifact you control."
Spawn of Mayhem	4	190	CS	2.878787879	1	Spectacle [1][Black][Black] (You may cast this spell for its spectacle cost rather than its mana cost if an opponent lost life this turn.)[LINE BREAK]Flying| trample[LINE BREAK]At the beginning of your upkeep| Spawn of Mayhem deals 1 damage to each player. Then if you have 10 or less life| put a +1/+1 counter on Spawn of Mayhem.
Trostani Discordant	4	179	CFP	2.418918919	0	Other creatures you control get +1/+1.[LINE BREAK]When Trostani Discordant enters the battlefield| create two 1/1 white Soldier creature tokens with lifelink.[LINE BREAK]At the beginning of your end step| each player gains control of all creatures they own.
Nullhide Ferox	4	148	CF	3.020408163	0	Hexproof[LINE BREAK]You can't cast noncreature spells.[LINE BREAK][2]: Nullhide Ferox loses all abilities until end of turn. Any player may activate this ability.[LINE BREAK]If a spell or ability an opponent controls causes you to discard Nullhide Ferox| put it onto the battlefield instead of putting it into your graveyard.
Ajani| Adversary of Tyrants	4	142	CP	1.543478261	1	+1: Put a +1/+1 counter on each of up to two target creatures.[LINE BREAK]â˜2: Return target creature card with converted mana cost 2 or less from your graveyard to the battlefield.[LINE BREAK]â˜7: You get an emblem with "At the beginning of your end step| create three 1/1 white Cat creature tokens with lifelink."

Figure 11.1: Screenshot of "Best Cards" in "Best_Cards--with_Inventory.xlsm"

Inventory

Created: 03/06/19
Updated: 04/18/19

Mana Type	Name	Inventory	Mana Type	Name	Inventory	Mana Type	Name	Inventory	Mana Type	Name	Inventory	Mana Type	Name	Inventory	Mana Type	Name	Inventory	Mana Type	Name	Inventory
0	Detection Tower	3	0	Forest	24	0	Island	24	0	Mountain	24	0	Plains	24	0	Swamp	24	0	Selesnya Guildgate	4	0	Highland Lake	4
	Field of Ruin	2	1	Adventurous Impulse	2	1	Arcane Flight	1	1	Crash Through	1	1	Ajani's Welcome	4	1	Blessing of Belzenlok	1		Sunpetal Grove	1		Izzet Guildgate	4
	Gateway Plaza	2		Commune with Dinosaurs	1		Benthic Biomancer	1		Deface	4		Arrester's Zeal	3		Demonic Vigor	3		Tranquil Expanse	4		Steam Vents	1
	Plaza of Harmony	1		Enter the Unknown	2		Curious Obsession	4		Fanatical Firebrand	2		Blazing Hope	1		Diregraf Ghoul	2	2	Conclave Guildmage	1		Sulfur Falls	1
	Reliquary Tower	1		Jade Bearer	4		Dazzling Lights	2		Fervent Strike	2		Charge	1		Duress	3		Emmara| Soul of the Accord	2	2	Beamsplitter Mage	3
	Rupture Spire	2		Jungle Delver	1		Dive Down	1		Ghitu Lavarunner	3		Dauntless Bodyguard	1		Fungal Infection	3		Siegehorn Ceratops	3		League Guildmage	1
	Unknown Shores	1		Kumena's Speaker	3		Gearsmith Prodigy	4		Goblin Motivator	1		Demotion	2		Grasping Scoundrel	3		Shanna| Sisay's Legacy	1	3	Adeliz| the Cinder Wind	1
	Zhalfirin Void	1		Llanowar Elves	4		Maximize Altitude	2		Rigging Runner	1		Gird for Battle	2		Hired Poisoner	1		Vernadi Shieldmate	1		Beacon Bolt	1
1	Fountain of Renewal	4		Might of the Masses	1		Mistcaller	3		Rile	1		Haazda Marshal	2		March of the Drowned	1	3	Centaur Peacemaker	4		Enigma Drake	3
	Navigator's Compass	1		Open the Gates	4		Pteramander	2		Shivan Fire	3		Healer's Hawk	2		Necrotic Wound	1		Knight of Autumn	2		Storm Fleet Sprinter	1
	Scrabbling Claws	1		Pelt Collector	2		Sea Legs	2		Shock	4		Hunted Witness	2		Nightmare's Thirst	2		Satyr Enchanter	4	4	Marauding Looter	1
	Screaming Shield	3		Portcullis Vine	2		Shore Keeper	1		Skirk Prospector	1		Leonin Vanguard	4		Pilfering Imp	1		Selesnya Locket	3	5	Ral| Izzet Viceroy	1
	Sentinel Totem	1		Prey Upon	1		Siren Stormtamer	4		Smelt	2		Moment of Triumph	2		Skulduggery	2	4	Conclave Cavalier	2	6	Invert / Invent (Invent)	1
	Silent Dart	1		Saruli Caretaker	4		Spell Pierce	2		Spear Spewer	3		Novice Knight	4		Stitcher's Supplier	3		Sumala Woodshaper	2			
	Sparring Construct	1		Stony Strength	3	2	Aether Tunnel	1		Storm Strike	3		Resolute Watchdog	1		Thirsting Shade	3	5	Join Shields	3			
	Wand of Vertebrae	1		Wall of Vines	4		Anticipate	2		Thud	2		Righteous Blow	1		Vampire Neonate	2		Rosemane Centaur	1			
2	Amulet of Safekeeping	2	2	Aggressive Urge	2		Blink of an Eye	3		Tin Street Dodger	1		Rustwing Falcon	4		Vicious Rumors	2		Camaraderie	3			
	Azor's Gateway	1		Cherished Hatchling	1		Chart a Course	2		Torch Courier	1		Sheltering Light	1	2	Abnormal Endurance	3	6	Flower // Flourish (Flourish)	3			
	Diamond Mare	4		Deeproot Champion	1		Departed Deckhand	4		Warlord's Fury	2		Sky Tether	1		Bankrupt in Blood	2	X	March of the Multitudes	1			
	Field Creeper	4		Deeproot Elite	2		Diligent Excavator	1	2	Cavalcade of Calamity	3		Skymarcher Aspirant	1		Bladebrand	2		Wordsoul Colossus	1			
	Glaive of the Guildpact	1		Deeproot Warrior	1		Disperse	4		Dismissive Pyromancer	2		Snubhorn Sentry	1		Cast Down	1						
	Glass of the Guildpact	1		Devkarin Dissident	1		Drowned Secrets	1		Doublecast	4		Tactical Advantage	3		Child of Night	1						
	Gleaming Barrier	1		Druid of the Cowl	4		Enhanced Surveillance	3		Electrostatic Field	1		Take Heart	1		Costly Plunder	2						
	Jousting Lance	1		Dryad Greenseeker	1		Essence Capture	4		Feral Maaka	4		Twilight Panther	2		Dinosaur Hunter	1						
	Marauder's Axe	2		Elvish Clancaller	2		Essence Scatter	1		Frenzied Rage	2	2	Adamant Will	1		Dire Fleet Poisoner	1						
	Millstone	3		Gaea's Blessing	1		Faerie Duelist	2		Goblin Cratermaker	3		Adanto Vanguard	2		Doomed Dissenter	4						
	Pillar of Origins	1		Gift of Growth	1		Favorable Winds	3		Goblin Instigator	4		Aegis of the Heavens	4		Infernal Scarring	4						
	Rogue's Gloves	1		Gift of Strength	4		Ghostform	1		Goblin Locksmith	1		Ajani's Pridemate	3		Kitesail Freebooter	2						
	Shield of the Realm	2		Giltgrove Stalker	2		Kitesail Corsair	4		Goblin Trailblazer	1		Baffling End	1		Macabre Waltz	2						
	Suspicious Bookcase	1		Grappling Sundew	1		Metamorphic Alteration	1		Gravel-Hide Goblin	4		Bishop's Soldier	4		Oathsworn Vampire	2						

Figure 11.2: Screenshot of "Inventory" in "Best_Cards--with_Inventory.xlsm"

```
'Entering_Inv_into_Best_Cards.bas
'
'This program enters card inventories into my organized MTG Arena inventory
'into a column in the best-cards database.
'
'Created: ??/??/2019
'Updated: 04/14/19
'
'Inputs: Best Cards worksheet, Inventory worksheet
'Dependencies: Microsoft Scripting Runtime
'Outputs: Information to fill in inventory column in Best Cards worksheet


Sub Enter_Inv_into_Best_Cards()


    'Specify the height of the header in the "Inventory" worksheet
    'in terms of number of rows.
    Dim header_height As Integer: header_height = 7

    'Create a dictionary with the column letters corresponding to columns of names as keys
    'and the numbers of names in the columns of names as items.
    Dim col_lets_and_nums_names As New Scripting.Dictionary
    col_lets_and_nums_names.Add Key:="B", Item:=Sheets("Inventory").Range("B" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="E", Item:=Sheets("Inventory").Range("E" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="H", Item:=Sheets("Inventory").Range("H" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="K", Item:=Sheets("Inventory").Range("K" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="N", Item:=Sheets("Inventory").Range("N" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="Q", Item:=Sheets("Inventory").Range("Q" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="U", Item:=Sheets("Inventory").Range("U" & Rows.Count).End(xlUp).Row - header_height
    col_lets_and_nums_names.Add Key:="Y", Item:=Sheets("Inventory").Range("Y" & Rows.Count).End(xlUp).Row - header_height

    'Create a dictionary with the column letters corresponding to columns of names as keys
    'and the column letters corresponding to columns of inventories as items.
    Dim name_lets_and_inv_lets As New Scripting.Dictionary
    name_lets_and_inv_lets.Add Key:="B", Item:="C"
    name_lets_and_inv_lets.Add Key:="E", Item:="F"
    name_lets_and_inv_lets.Add Key:="H", Item:="I"
    name_lets_and_inv_lets.Add Key:="K", Item:="L"
    name_lets_and_inv_lets.Add Key:="N", Item:="O"
    name_lets_and_inv_lets.Add Key:="Q", Item:="R"
    name_lets_and_inv_lets.Add Key:="U", Item:="V"
    name_lets_and_inv_lets.Add Key:="Y", Item:="Z"


    'For each card in the best-cards database...
    Dim i As Integer
    For i = 2 To 1226

        'Set the inventory for the present card to 0.
        Sheets("Best Cards").Range("F" & i).Value = 0

    Next i


    'For each column of names...
    Dim j As Integer
    For i = 1 To col_lets_and_nums_names.Count

        'For each card in the present column...
        For j = 1 To col_lets_and_nums_names.Items(i - 1)

            'For each card in the best-cards database...
            For k = 2 To 1226

                'If the present card in the best-cards database is the same as the present card in the present column of names...
                If StrComp(Sheets("Best Cards").Range("A" & k).Value, Sheets("Inventory").Range(col_lets_and_nums_names.Keys(i - 1) & header_height + j).Value) = 0 Then

                    'Set the inventory for the present card in the best-cards database to the inventory of the present card in the present column.
                    Sheets("Best Cards").Range("F" & k).Value = Sheets("Inventory").Range(name_lets_and_inv_lets(col_lets_and_nums_names.Keys(i - 1)) & header_height + j).Value

                    Exit For 'k

                End If

            Next k

        Next j

    Next i

End Sub
```

Figure 11.3: Screenshot of "Enter_Inv_into_Best_Cards.bas"

2.12. Finding Average Win Ratios of Decks with Mana Types in Specific Groupings

Return to Tasks Checklist

I wrote the below Python program to find the average win ratios of decks with mana types in specific groupings. For example, I found the average win ratios of decks with mana types in the grouping ("C", "I", "M", "CI", "CM", "CIM") to be 53.1 percent. I can use this information to make an educated guess that if I play a deck with mana types in the above grouping I will win more often than if I play a deck with mana types in any other grouping. I will definitely filter the best-cards database with inventory into a strong deck with mana types corresponding to a flavor that I enjoy playing and corresponding to a grouping with a high average win ratio.

My program requires the path to the condensed cards database. My program requires the path to the name x ratio database. My program relies on the pandas Python library to read the condensed cards database and name x ratio database into dataframes and to manipulate the dataframes. My program outputs mana type groupings and corresponding average win ratios to this notebook.

In [105]:
```
# Finding_Average_Win_Ratios_of_Decks_with_Mana_Types_in_Specific_Groupings.py
#
# This program ouputs mana-type groupings and corresponding average win ratios
to this notebook.
#
# Created: 03/??/19 by Tom Lever
# Updated: 04/02/19 by Tom Lever
#
# Inputs: Condensed_Cards_Database.csv, Name_x_Ratio_Database.csv
# Dependencies: pandas
# Outputs: A table of mana-type groupings and average win ratios


# Allow use of the pd.read_csv method and manipulation of imported dataframes.
import pandas as pd


# Read the condensed cards database into a dataframe, 1546 x 11, with index co
lumn and header row additional.
condensed_cards_database = pd.read_csv("Condensed_Cards_Database.csv", index_c
ol=0)

# Enter the names column of the condensed cards database into a list, 1546 x
 1.
names_in_CCD = condensed_cards_database.index.tolist()

# Enter the mana-types column of the condensed cards database into a list, 154
6 x 1.
mana_types = condensed_cards_database["Mana Type"].tolist()

# Create a table of names and mana types, 1546 x 1, with index column addition
al.
names_and_mana_types = pd.DataFrame(mana_types, columns=["Mana Type"], index=n
ames_in_CCD)


# Enter the name x ratio database into a dataframe, 1232 x 2271, with index co
lumn and header row additional.
name_x_ratio_database = pd.read_csv("./Data_With_Ravnica_Allegiance/Name_x_Rat
io_Database.csv", header=None, index_col=0)
name_x_ratio_database.columns = name_x_ratio_database.iloc[0].rename("Name")
del name_x_ratio_database.index.name
name_x_ratio_database = name_x_ratio_database.iloc[1:]


# Create a name and mana type x win / loss ratio dataframe.
# Please note that the resulting dataframe is 1225 x 2,272, with index column
 and header row additional.
# Angelic Reward, Confront the Assault, Inspiring Commander, Spiritual Guardia
n, Tactical Advantage, Angelic Guardian,
# and Rampaging Brontodon are not in Magic the Gathering's Gatherer database,
 and are only in MTG Arena.
# The additional column is of course the mana-type column.
name_mana_type_x_ratio_database = name_x_ratio_database.merge(names_and_mana_t
ypes, left_index=True, right_index=True)
```

```
# Enter the mana-type column into a series, 1225 x 1, with index column additi
onal.
mana_type_column = name_mana_type_x_ratio_database["Mana Type"]


#########################################################################
#######################
# Create a list of long strings, each containing all of the mana costs of all
 of the cards in a deck.
#########################################################################
#######################

list_of_long_strings = []

# For each deck in the name and mana type x win / loss ratio database...
for j in range(0, name_mana_type_x_ratio_database.shape[1]-1):

    # Create a series of card names and numbers in the present deck.
    card_nums = name_mana_type_x_ratio_database.iloc[:, j]
    card_nums = card_nums[card_nums != 0]

    # Add the mana type for each card in the present deck to a long string.
    long_string = ""
    for name in card_nums.index:
        long_string += str(mana_type_column[name])

    # Add the long string for the present deck to a list of long strings for a
ll decks.
    list_of_long_strings.append(long_string)


###################################
# Create a list of deck mana types.
###################################

list_of_deck_mana_types = []

# For each long string containing all of the mana costs of all of the cards in
a deck...
for long_string in list_of_long_strings:

    # Create a string of deck mana types.
    deck_mana_types = ""
    if "C" in long_string:
        deck_mana_types += "C"
    if "F" in long_string:
        deck_mana_types += "F"
    if "I" in long_string:
        deck_mana_types += "I"
    if "M" in long_string:
        deck_mana_types += "M"
    if "P" in long_string:
        deck_mana_types += "P"
    if "S" in long_string:
        deck_mana_types += "S"

    # Add the string of deck mana types to a list of deck mana types for all d
```

```
ecks.
    list_of_deck_mana_types.append(deck_mana_types)

# Create a table of deck mana types and win ratios, 2271 x 1, with index colum
n additional.
mana_types_and_win_ratios = pd.DataFrame(list_of_deck_mana_types, columns=["Ma
na Type"], index=name_x_ratio_database.columns.values.tolist())

print("Please see below for average win ratios for decks with mana types in ce
rtain groupings.")

# For a mana-type grouping of ("C")...
mana_type_groupings = [("C")]
for grouping in mana_type_groupings:

    # Find the average win ratio for all decks with mana types in the present
 mana-type grouping.
    ave_win_ratio = np.mean(mana_types_and_win_ratios[mana_types_and_win_ratio
s["Mana Type"] == grouping[0]].index.values.astype(np.float))

    groupings.append(grouping)
    average_win_ratios.append(ave_win_ratio)

    # Print the present grouping and the average win ratio.
    print(str(grouping) + ": %.1f" % ave_win_ratio + "%")

# For each "mono-colored" mana-type grouping...
mana_type_groupings = [("C", "F", "CF"),
                       ("C", "I", "CI"),
                       ("C", "M", "CM"),
                       ("C", "P", "CP"),
                       ("C", "S", "CS")]

for grouping in mana_type_groupings:

    mask = mana_types_and_win_ratios["Mana Type"] == grouping[0]
    for i in range(1, len(grouping)):
        mask = mask | (mana_types_and_win_ratios["Mana Type"] == grouping[i])
    ave_win_ratio = np.mean(mana_types_and_win_ratios[mask].index.values.astyp
e(np.float))

    groupings.append(grouping)
    average_win_ratios.append(ave_win_ratio)

    print(str(grouping) + ": %.1f" % ave_win_ratio + "%")

mana_type_groupings = [("C", "F", "I", "FI", "CFI"),
                       ("C", "F", "M", "FM", "CFM"),
                       ("C", "F", "P", "FP", "CFP"),
                       ("C", "F", "S", "FS", "CFS"),
                       ("C", "I", "M", "IM", "CIM"),
                       ("C", "I", "P", "IP", "CIP"),
                       ("C", "I", "S", "IS", "CIS"),
                       ("C", "M", "P", "MP", "CMP"),
                       ("C", "M", "S", "MS", "CMS"),
                       ("C", "P", "S", "PS", "CPS")]
```

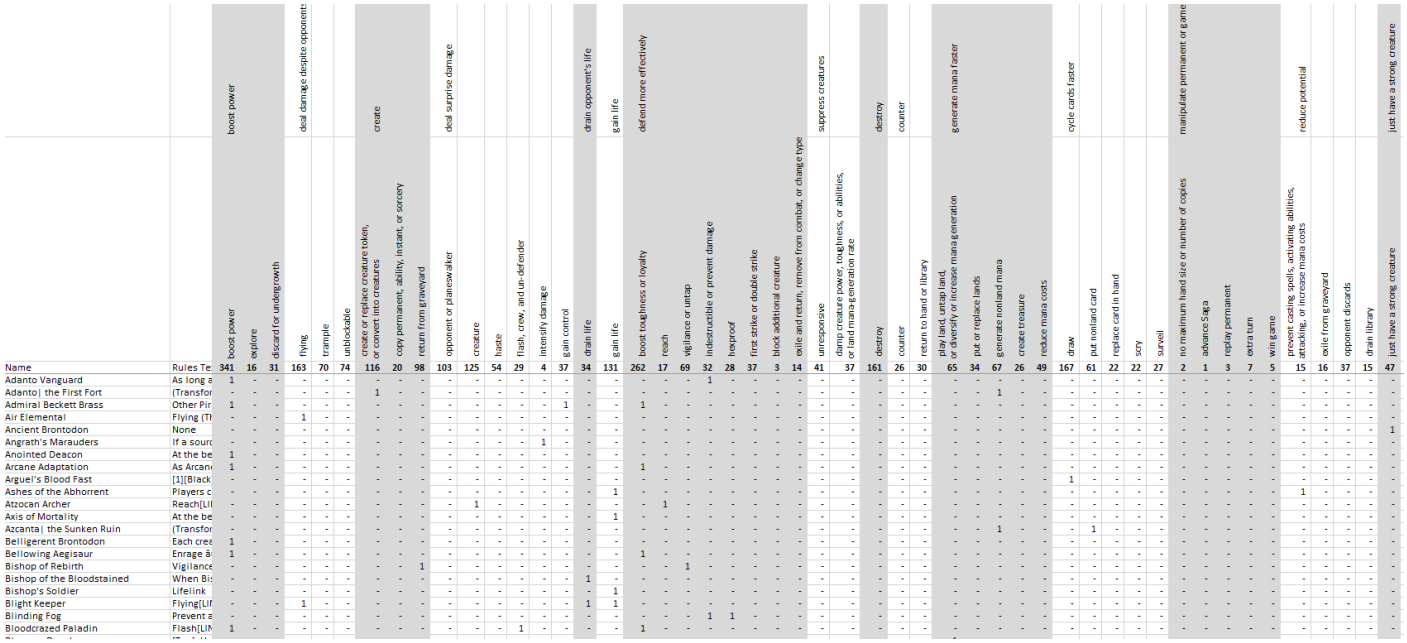
```
for grouping in mana_type_groupings:

    mask = mana_types_and_win_ratios["Mana Type"] == grouping[0]
    for i in range(1, len(grouping)):
        mask = mask | (mana_types_and_win_ratios["Mana Type"] == grouping[i])
    ave_win_ratio = np.mean(mana_types_and_win_ratios[mask].index.values.astyp
e(np.float))

    groupings.append(grouping)
    average_win_ratios.append(ave_win_ratio)

    print(str(grouping) + ": %.1f" % ave_win_ratio + "%")
```

```
Please see below for average win ratios for decks with mana types in certain
groupings.
C: nan%
('C', 'F', 'CF'): 46.0%
('C', 'I', 'CI'): 47.0%
('C', 'M', 'CM'): 48.4%
('C', 'P', 'CP'): 50.6%
('C', 'S', 'CS'): 50.3%
('C', 'F', 'I', 'FI', 'CFI'): 50.0%
('C', 'F', 'M', 'FM', 'CFM'): 51.6%
('C', 'F', 'P', 'FP', 'CFP'): 51.3%
('C', 'F', 'S', 'FS', 'CFS'): 49.1%
('C', 'I', 'M', 'IM', 'CIM'): 53.1%
('C', 'I', 'P', 'IP', 'CIP'): 48.4%
('C', 'I', 'S', 'IS', 'CIS'): 50.0%
('C', 'M', 'P', 'MP', 'CMP'): 49.8%
('C', 'M', 'S', 'MS', 'CMS'): 49.8%
('C', 'P', 'S', 'PS', 'CPS'): 49.2%
```

2.13. Developing a Rules Text Categories Database

Return to Tasks Checklist

I developed a rules text categories database. My rules text categories database classifies every card in the condensed cards database as belonging to one or more of fifty categories based on the content of the card's rules text. A categories of interest database will be created using this database. Please see below screenshot for what my database looks like.



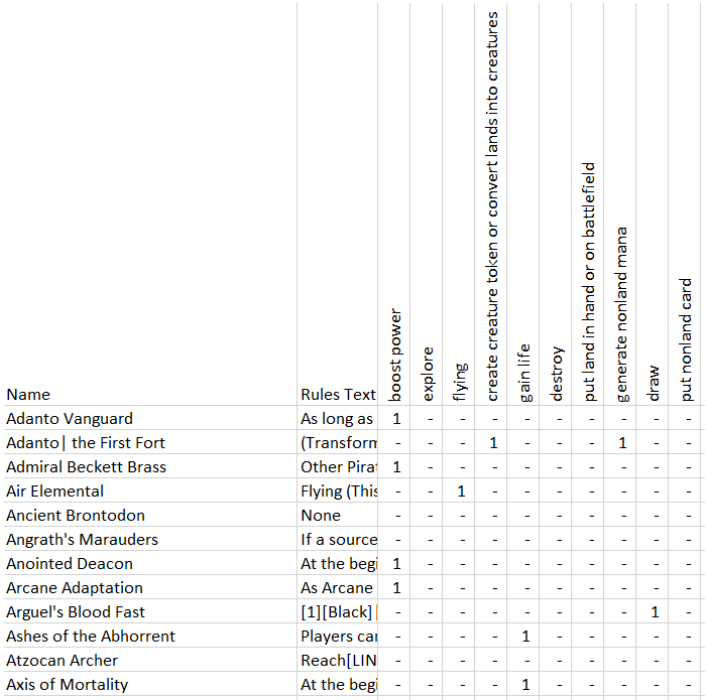
		boost power			deal damage despite opponents			create			deal surprise damage						drain opponent's life	gain life	defend more effectively							suppress creatures			destroy	counter		generate mana faster					cycle cards faster					manipulate permanent or game					reduce potential				just have a strong creature
		boost power	explore	discard for undergrowth	flying	trample	unblockable	create or replace creature token, or convert into creatures	copy permanent, ability, instant, or sorcery	return from graveyard	opponent or planeswalker	creature	haste	flash, crew, and un-defender	intensify damage	gain control	drain life	gain life	boost toughness or loyalty	reach	vigilance or untap	indestructible or prevent damage	hexproof	first strike or double strike	block additional creature	exile and return, remove from combat, or change type	unresponsive	damp creature power, toughness, or abilities, or land mana generation rate	destroy	return to hand or library	play land, untap land, or diversify or increase mana generation	put or replace lands	generate nonland mana	create treasure	reduce mana costs	draw	put nonland card	replace card in hand	scry	surveil	no maximum hand size or number of copies	advance Saga	replay permanent	extra turn	win game	prevent casting, spells, activating abilities, attacking, or increase mana costs	exile from graveyard	opponent discards	drain library	just have a strong creature
Name	Rules Te	341	16	31	163	70	74	116	20	98	103	125	54	29	4	37	34	131	262	17	69	32	28	37	3	14	41	37	161	26	30	65	34	67	26	49	167	61	22	22	27	2	1	3	7	5	15	16	37	15	47
Adanto Vanguard	As long a	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Adanto	the First Fort	(Transfor	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Admiral Beckett Brass	Other Pir	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Air Elemental	Flying (Th	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Ancient Brontodon	None	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
Angrath's Marauders	If a sourc	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Anointed Deacon	At the be	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Arcane Adaptation	As Arcan	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Arguel's Blood Fast	[1][Black	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
Ashes of the Abhorrent	Players c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-
Atzocan Archer	Reach[Lil	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Axis of Mortality	At the be	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Azcanta	the Sunken Ruin	(Transfor	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
Belligerent Brontodon	Each cree	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Bellowing Aegisaur	Enrage a	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Bishop of Rebirth	Vigilance	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Bishop of the Bloodstained	When Bis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Bishop's Soldier	Lifelink	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Blight Keeper	Flying[Lif	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Blinding Fog	Prevent a	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Bloodcrazed Paladin	Flash[Lin	1	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 13: Screenshot of "Rules_Text_Categories.csv"

2.14. Developing Categories of Interest Database

Return to Tasks Checklist

I developed a categories of interest database by extracting columns of interest from the rules text categories database and reformatting. My categories of interest database classifies every card in the condensed cards database as belonging, or not, to one or more of the categories of interest, based on the content of the card's rules text. When I only had half of the rules text categories database filled in, I used the below machine-learning techniques to fill in the second half of the appropriate categories of interest columns. When Wizards of the Coast publishes a new card set, I will be able to use the below machine-learning techniques to guess at how the new cards may be assigned to my fifty categories. Regardless of whether or not I am using my machine-learning program to fill in the categories of interest database, I will create a filled-in categories of interest database using the categories of interest database. Please see below screenshot for what my categories of interest database looks like.



Name	Rules Text	boost power	explore	flying	create creature token or convert lands into creatures	gain life	destroy	put land in hand or on battlefield	generate nonland mana	draw	put nonland card
Adanto Vanguard	As long as	1	-	-	-	-	-	-	-	-	-
Adanto| the First Fort	(Transforn	-	-	-	1	-	-	-	1	-	-
Admiral Beckett Brass	Other Pira	1	-	-	-	-	-	-	-	-	-
Air Elemental	Flying (This	-	-	1	-	-	-	-	-	-	-
Ancient Brontodon	None	-	-	-	-	-	-	-	-	-	-
Angrath's Marauders	If a source	-	-	-	-	-	-	-	-	-	-
Anointed Deacon	At the begi	1	-	-	-	-	-	-	-	-	-
Arcane Adaptation	As Arcane	1	-	-	-	-	-	-	-	-	-
Arguel's Blood Fast	[1][Black]	-	-	-	-	-	-	-	-	1	-
Ashes of the Abhorrent	Players ca	-	-	-	-	1	-	-	-	-	-
Atzocan Archer	Reach[LIN	-	-	-	-	-	-	-	-	-	-
Axis of Mortality	At the begi	-	-	-	-	1	-	-	-	-	-

Figure 14: Screenshot of "Categories_of_Interest.csv"

2.15. Filling Categories of Interest Database

Return to Tasks Checklist

2.15.1. Task, Software, and Result

I wrote the below Python program to use a machine learning model to fill in my categories of interest database. A filtered filled-in categories database will be created using this database. The beginning of machine-generated filled-in categories database of course looks the same as the categories of interest database.

My program requires the path to the categories of interest database. My program relies on the pandas Python library to read the categories of interest database into a dataframe. My program relies on sklearn.pipeline.Pipeline, sklearn.feature_extraction.text.CountVectorizer, sklearn.feature_extraction.text.TfidfTransformer, and sklearn.linear_model.SGDClassifier classes to develop the machine-learning model. My program relies on numpy Python package to find an model-prediction accuracy for each category. My program writes a filled-in categories database to a CSV file.

In [1]:
```
# Filling_Categories_of_Interest_Database.py
#
# This program using a machine-learning model to fill in my categories of inte
rest database.
#
# Created: 04/02/19 by Tom Lever
# Updated: 04/17/19 by Tom Lever
#
# Inputs: Categories_of_Interest.csv
# Dependencies: pandas, numpy, sklearn.pipeline.Pipeline, sklearn.feature_extr
action.text.CountVectorizer,
# sklearn.feature_extraction.text.TfidfTransformer, sklearn.linear_model.SGDCl
assifier
# Outputs: Filled_In_Categories_Database.csv


# Allow use of the pandas.read_csv method.
import pandas as pd

# Allow creation of a sklearn.pipeline.Pipeline class instance.
from sklearn.pipeline import Pipeline

# Allow the pipeline's fit method to use the sklearn.feature_extraction.text.C
ountVectorizer class.
from sklearn.feature_extraction.text import CountVectorizer

# Allow the pipeline's fit method to use the sklearn.feature_extraction.text.T
fidfTransformer class.
from sklearn.feature_extraction.text import TfidfTransformer

# Allow the pipeline's fit method to use the sklearn.linear_model.SGDClassifie
r class.
from sklearn.linear_model import SGDClassifier

# Allow use of the numpy.mean method.
import numpy as np


# Enter the categories_of_interest database into a dataframe.
categories_database = pd.read_csv("./Data_With_Ravnica_Allegiance/Categories_o
f_Interest.csv", header=0, index_col=0)


##############################################################################
##################################
# Determine the accuracy of a machine-learning model that will fill in the maj
ority of the categories database.
##############################################################################
##################################

categories = []
accuracies = []

# Enter the column of rules texts into a list.
list_of_rules_texts = categories_database["Rules Text"].tolist()
```

```
# Designate some of the rules texts as training rules texts.
training_rules_texts = list_of_rules_texts[0:950]

# Designate some of the rules texts as test rules texts
# for determining the accuracy of the model for each category.
test_rules_texts = list_of_rules_texts[950:1150]

# For each category in the categories database...
for category in categories_database.columns.values.tolist()[1:]:

    # Specify a column of training values corresponding to the training rules
 texts.
    training_values = categories_database[category].tolist()[0:950]

    # Develop the machine-learning model.
    text_clf_svm = Pipeline([("vect", CountVectorizer()), ("tfidf", TfidfTrans
former()), ("clf-svm", SGDClassifier(loss="hinge", penalty="l2", alpha=1e-3, n
_iter=5, random_state=42))])
    text_clf_svm = text_clf_svm.fit(training_rules_texts, training_values)

    # Predict whether each test rules text belongs to the present category.
    predicted_svm_values = text_clf_svm.predict(test_rules_texts)

    # Find the actual values for whether each test rules text belongs to the p
resent category.
    actual_values = categories_database[category].tolist()[950:1150]

    categories.append(category)
    accuracies.append(np.mean(predicted_svm_values == actual_values))

print("The following table presents the accuracy of this program's machine-lea
rning model")
print("in predicting whether a card in the categories database belongs to a gi
ven category.")
print("This model will be used to fill in the majority of the categories datab
ase.")
print('As you can see, the model got an "A" for each category.')
print(pd.DataFrame(accuracies, columns=["Accuracy"], index=categories))


###################################
# Fill in the categories database.
###################################

# Enter the column of card names into a list.
names = categories_database.index.tolist()

filled_in_categories_database = pd.DataFrame(categories_database["Rules Text"]
.tolist(), columns=["Rules Text"], index=names)

# Specify the rules texts needing corresponding predicted values.
test_rules_texts = list_of_rules_texts[950:]

for category in categories_database.columns.values.tolist()[1:]:

    # Re-specify each column of training values corresponding to the training
 rules texts.
```

```
    training_values = categories_database[category].tolist()[0:950]

    # Redevelop the model.
    text_clf_svm = text_clf_svm.fit(training_rules_texts, training_values)

    # Predict the values required to fill in the column corresponding to the p
resent category.
    predicted_svm_values = text_clf_svm.predict(test_rules_texts)

    # Define the full column of values corresponding to the present category.
    theoretical_values = training_values + predicted_svm_values.tolist()

    # Refresh the column in the categories database corresponding to the prese
nt category.
    filled_in_categories_database[category] = theoretical_values

# Print the filled in database.
filled_in_categories_database.to_csv("./Data_With_Ravnica_Allegiance/Filled_In
_Categories_Database.csv")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gr
adient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and
will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
```

```
The following table presents the accuracy of this program's machine-learning model
in predicting whether a card in the categories database belongs to a given category.
This model will be used to fill in the majority of the categories database.
As you can see, the model got an "A" for each category.
                                                     Accuracy
boost power                                             0.940
explore                                                 1.000
 flying                                                 0.980
 create creature token or convert lands into cr...      0.980
 gain life                                              1.000
 destroy                                                0.970
 generate nonland mana                                  0.975
 draw                                                   0.950
 put nonland card                                       0.975

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:152: DeprecationWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and tol instead.
  DeprecationWarning)
```

2.15.2. Beginning to Think About Teaching a Computer to Come Up with My Categories

I wrote the below program and Grammar.fcfg file to acknowledge some ideas I had regarding teaching a computer to come up with my categories in the rules text categories database. Initially, I experimented with finding core transitive verb phrases by replacing "[LINE BREAK]" strings with " " characters and removing conjunctional, adverbial, and prepositional phrases. I discovered that it was unwieldy to use the re Python package to search for strings that began with pivotal words and ended with punctuation. I then experimented with using Python's Natural Language Tool Kit to parse rules-text sentences into user-defined grammatical structures and return core transitive verb phrases. I successfully parsed a few dozen rules-text sentences, but soon ran into difficulty handling subordinate clauses and sentences where important categorizing information was imbedded in prepositional phrases. Finally, as depicted in the below program and Grammar.fcfg file, I played with the idea of making sentence parsing easier by including more grammar information in the sentences by translating the sentences into a non-English language with more verb conjugations and prepositional phrases instead of attributive nouns. In practice, doing so turned out for me to be an ambiguous and painstaking process. All in all, I am glad I defined by hand my 50 categories in the rules text categories database, especially because my categories changed as I viewed more rules texts, came up with more categories, and grouped categories together.

My program relies on Python's Natural Language Tool Kit to load a grammatic rule set that I wrote and to parse sentences into grammatical structures based on that rule set.

In [1]:
```
import nltk

sentence = "as_long_as [NAME] is attacking| it gets [BOOST]."
sentence = "mientras [NAME] está atacando| él consigue [BOOST]."

sentence = sentence.replace("|", "")
sentence = sentence.replace(".", "")
sentence = sentence.split(" ")

chart_parser = nltk.load_parser("Grammar.fcfg")
for tree in chart_parser.parse(sentence):
    print(str(tree))
```

```
(Sentence[struct='fronted']
  (Conjunctive_Phrase[]
    (Conjunction[] mientras)
    (Sentence[struct='simple', tense='pres_prog']
      (Noun[] [NAME])
      (Verb_Phrase[tense='pres_prog']
        (Verb[] está)
        (Participle[] atacando))))
  (Sentence[struct='simple', tense='pres']
    (Noun[] él)
    (Verb_Phrase[tense='pres'] (Verb[] consigue) (Noun[] [BOOST]))))
```

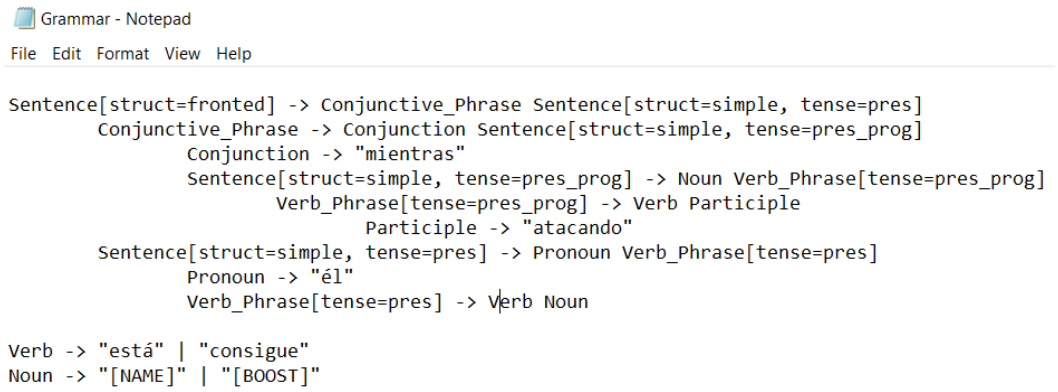


Figure 15: Screenshot of "Grammar.fcfg"

2.16. Filtering Filled-In Categories Database

Return to Tasks Checklist

I wrote the below Python program to filter my filled-in categories database by mana type and availability. This database will be used to find the highest average win ratios for all combinations of categories in the filled-in categories database. Please see below screenshot of the filtered filled-in categories database.

My program requires the path to the condensed cards database. My program requires the path to the best-cards database with inventory. My program of course requires the path to the filled-in categories database. My program relies on the pandas Python library to read the databases into dataframes and to manipulate the dataframes. My program outputs the filtered filled-in categories database to a CSV file.

	Rules Text	boost power	explore	flying	create creature token or convert lands into	gain life	destroy	generate nonland mana	draw	put nonland card	Mana Type	Inventory
Adanto Vanguard	As long as	1	-	-	-	-	-	-	-	-	CP	2
Ancient Brontodon	None	-	-	-	-	-	-	-	-	-	CF	2
Bishop's Soldier	Lifelink	-	-	-	-	1	-	-	-	-	CP	4
Bright Reprisal	Destroy ta	-	-	-	-	-	1	-	1	-	CP	1
Carnage Tyrant	This spell c	-	-	-	-	-	-	-	-	-	CF	1
Colossal Dreadmaw	Trample (T	-	-	-	-	-	-	-	-	-	CF	2
Commune with Dinosaurs	Look at th	-	-	-	-	-	-	-	-	1	F	1
Crushing Canopy	Choose on	-	-	-	-	-	1	-	-	-	CF	3
Deeproot Champion	Whenever	1	-	-	-	-	-	-	-	-	CF	1
Deeproot Warrior	Whenever	1	-	-	-	-	-	-	-	-	CF	1
Field of Ruin	[Tap]: Add	-	-	-	-	-	1	1	-	-	C	2
Forest	G	-	-	-	-	-	-	-	-	-	F	24
Gilded Sentinel	None	-	-	-	-	-	-	-	-	-	C	4
Goring Ceratops	Double str	-	-	-	-	-	-	-	-	-	CP	1
Imperial Aerosaur	Flying[LINE	1	-	1	-	-	-	-	-	-	CP	3
Inspiring Cleric	When Insp	-	-	-	-	1	-	-	-	-	CP	4
Ixalan's Binding	When Ixala	-	-	-	-	-	1	-	-	-	CP	1
Ixalli's Diviner	When Ixall	-	1	-	-	-	-	-	-	-	CF	1
Kinjalli's Sunwing	Flying[LINE	-	-	1	-	-	-	-	-	-	CP	2
Kumena's Speaker	Kumena's	1	-	-	-	-	-	-	-	-	F	3
Merfolk Branchwalker	When Mer	-	1	-	-	-	-	-	-	-	CF	4

Figure 16: Screenshot of Filtered Filled-In Categories Database

In [1]:
```
# Filtering_Filled_In_Categories_Database.py
#
# This program filters my filled-in categories database by mana type and avail
ability.
#
# Created: 04/02/19 by Tom Lever
# Updated: 04/08/19 by Tom Lever
#
# Inputs: Condensed_Cards_Database.csv, Best_Cards--with_Inventory.csv, Filled
_In_Categories_Database.csv
# Dependencies: pandas
# Outputs: Filtered_Filled_In_Categories_Database.csv


import pandas as pd


# Enter the condensed cards database into a dataframe.
condensed_cards_database = pd.read_csv("./Data_With_Ravnica_Allegiance/Condens
ed_Cards_Database.csv")

# Create a table of card names and mana types from the condensed cards databas
e.
names_and_mana_types = pd.DataFrame(condensed_cards_database["Mana Type"].toli
st(), columns=["Mana Type"], index=condensed_cards_database["Name"].tolist())

# Filter the names and mana types table into a white-card names and mana types
table.
mask = names_and_mana_types["Mana Type"] == "C"
for mana_type in ["F", "P", "CF", "CP", "FP", "CFP", "Any"]:
    mask = mask | (names_and_mana_types["Mana Type"] == mana_type)

names_and_mana_types_filtered_by_mana_type = names_and_mana_types[mask]


# Enter the best cards database with inventory into a dataframe.
best_cards_database_with_inventory = pd.read_csv("./Data_with_Ravnica_Allegian
ce/Best_Cards--with_Inventory.csv", index_col=0).fillna(0)

# Create a table of card names and inventories from the best cards with invent
ory database.
names_and_inventories = pd.DataFrame(best_cards_database_with_inventory["Inven
tory"], index=best_cards_database_with_inventory.index)

# Filter the names and inventories table into an available names and inventori
es table.
is_available = names_and_inventories["Inventory"] > 0
names_and_inventories_filtered_by_inventory = names_and_inventories[is_availab
le]


# Enter the filled in categories database into a dataframe.
filled_in_categories_database = pd.read_csv("./Data_With_Ravnica_Allegiance/Fi
lled_In_Categories_Database.csv", header=0, index_col=0)

# Filter the filled in categories database by mana type and availability.
```

```
filtered_filled_in_categories_database = filled_in_categories_database.merge(n
ames_and_mana_types_filtered_by_mana_type, left_index=True, right_index=True)
filtered_filled_in_categories_database = filtered_filled_in_categories_databas
e.merge(names_and_inventories_filtered_by_inventory, left_index=True, right_in
dex=True)

# Write the filtered filled-in categories database to a CSV file.
filtered_filled_in_categories_database.to_csv("./Data_With_Ravnica_Allegiance/
Filtered_Filled_In_Categories_Database.csv")
```

2.17. Determining Highest Average Win / Loss Ratios for Combinations of Categories

Return to Tasks Checklist

I wrote the below Python program to determine the highest average win / loss ratio for each grouping of categories in the categories of interest database. For each possible number of unique cards in a deck that each happen to be in at least one category in a grouping, the win / loss ratios of all decks with that number of unique cards are averaged. The highest average win / loss ratio assigned to a grouping of categories is the highest average win / loss ratio among average win / ratios for different numbers of unique cards in categories. Please see below screenshot of my table of category combinations / groupings and corresponding highest average win / loss ratios.

My program requires the path to my name x ratio database. My program relies on the pandas Python library to read the name x ratio database into a dataframe. My program relies on the itertools.combinations class to create a list of all combinations of categories in the categories of interest database. My program relies on the numpy Python package to create storage for all average win / loss ratios for a given category grouping. My program outputs a table of category groupings and corresponding highest average win / loss ratios to a CSV file.

	Highest Average Ratio
('create creature token or convert lands into creatures', 'destroy', 'draw', 'put nonland card')	51.862
('create creature token or convert lands into creatures', 'destroy', 'draw')	51.821
('explore', 'create creature token or convert lands into creatures', 'destroy', 'draw', 'put nonland card')	51.592
('explore', 'create creature token or convert lands into creatures', 'destroy', 'draw')	51.590
('create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'put nonland card')	51.568
('boost power', 'explore', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana')	51.491
('boost power', 'explore', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana')	51.485
('create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana')	51.471
('explore', 'create creature token or convert lands into creatures', 'destroy', 'generate nonland mana', 'draw')	51.448
('boost power', 'explore', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'put nonland card')	51.433
('boost power', 'explore', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy')	51.424
('explore', 'create creature token or convert lands into creatures', 'destroy', 'generate nonland mana', 'draw', 'put nonland card')	51.418
('flying', 'destroy', 'draw')	51.409
('create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'draw', 'put nonland card')	51.344
('boost power', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana')	51.314
('flying', 'draw')	51.313
('boost power', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana')	51.308
('boost power', 'explore', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'put nonland card')	51.285
('boost power', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'put nonland card')	51.257
('explore', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana')	51.253
('explore', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'put nonland card')	51.248
('boost power', 'explore', 'create creature token or convert lands into creatures', 'generate nonland mana')	51.227
('boost power', 'explore', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'put nonland card')	51.226
('create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'draw')	51.200
('boost power', 'explore', 'flying', 'create creature token or convert lands into creatures', 'gain life', 'draw')	51.195
('flying', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'put nonland card')	51.190
('explore', 'create creature token or convert lands into creatures', 'gain life', 'destroy', 'generate nonland mana', 'draw', 'put nonland card')	51.189
('boost power', 'explore', 'flying', 'create creature token or convert lands into creatures', 'destroy', 'generate nonland mana', 'draw')	51.185

Figure 17: Screenshot of Category_Combinations_and_Ratios.csv

In [2]:
```
# Determining_Highest_Average_Win_Loss_Ratios_for_Combinations_of_Categories.p
y
#
# This program creates a table of highest average win / loss ratios
# for all combinations of categories in the filtered filled-in categories data
base.
# The table is sorted in descending order by highest average win / loss ratio.
#
# Created: 04/02/19 by Tom Lever
# Updated: 04/08/19 by Tom Lever
#
# Inputs: Name_x_Ratio_Database.csv
# Dependencies: pandas, itertools.combinations, numpy
# Outputs: Category_Combinations_and_Ratios.csv


# Allow use of the pandas.read_csv method.
import pandas as pd

# Allow creation of an itertools.combinations class instance.
from itertools import combinations

# Allow creation of a numpy ndarray of zeros.
import numpy as np


list_of_groupings = []
list_of_highest_average_ratios = []


# Enter the filtered filled in categories database into a dataframe.
filtered_filled_in_categories_database = pd.read_csv("./Data_With_Ravnica_Alle
giance/Filtered_Filled_In_Categories_Database.csv", index_col=0, header=0)

# Enter the name x ratio database into a dataframe, 1232 x 2271, with index co
lumn and header row additional.
name_x_ratio_database = pd.read_csv("./Data_With_Ravnica_Allegiance/Name_x_Rat
io_Database.csv", header=None, index_col=0)
name_x_ratio_database.columns = name_x_ratio_database.iloc[0].rename("Name")
del name_x_ratio_database.index.name
name_x_ratio_database = name_x_ratio_database.iloc[1:]


# For each possible number of categories in a group...
for i in range(1, len(filtered_filled_in_categories_database.columns[1:-2]) +
1):

    # Create a list of all possible groupings of categories.
    list_of_groupings_of_categories = list(combinations(filtered_filled_in_cat
egories_database.columns[1:-2], i))

    # For each grouping of categories...
    for grouping_of_categories in list_of_groupings_of_categories:

        # For each category, add the corresponding column from the filtered, f
illed-in categories database
```

```
        # to a name x category database.
        name_x_category_database = pd.DataFrame(filtered_filled_in_categories_
database[grouping_of_categories[0]])
        for j in range(1, len(grouping_of_categories)):
            name_x_category_database[grouping_of_categories[j]] = filtered_fil
led_in_categories_database[grouping_of_categories[j]]

        # Create a name x ratio and category database.
        name_x_ratio_and_category_database = name_x_ratio_database.merge(name_
x_category_database, left_index=True, right_index=True)

        # Filter the name x ratio and category database into a dataframe of ca
rds
        # with each card in at least one of the categories in the present grou
ping.
        mask = name_x_ratio_and_category_database[grouping_of_categories[0]] =
= 1
        for j in range(1, len(grouping_of_categories)):
            mask = mask | name_x_ratio_and_category_database[grouping_of_categ
ories[j]]
        filtered_name_x_ratio_and_category_database = name_x_ratio_and_categor
y_database[mask]

        # For each deck in the name x ratio database...
        numbers = []
        for j in range(0, name_x_ratio_database.shape[1]-1):

            # For the present deck, find all cards that are in at least one ca
tegory in the present grouping.
            cards = filtered_name_x_ratio_and_category_database.iloc[:, j]
            cards = cards[cards != 0]

            # For the present deck, record the total number of unique cards,
            # each of which is in at least one category in the present groupin
g.
            numbers.append(cards.shape[0])

        # Create a table of numbers of unique cards and win ratios.
        ratios = name_x_ratio_database.columns.values.tolist()
        numbers_and_ratios = pd.DataFrame(list(map(list, zip(*[numbers, ratios
]))), columns=["Number", "Ratio"])

        # For each possible number of unique cards...
        average_win_ratios = np.zeros(60)
        for j in range(0, 60):

            # Create a list of win ratios for decks with the present number of
unique cards.
            list_of_win_ratios_for_decks_with_j_unique_cards = [float(k) for k
in numbers_and_ratios[numbers_and_ratios["Number"] == 1]["Ratio"].tolist()]

            # If the list is not empty...
            if len(list_of_win_ratios_for_decks_with_j_unique_cards) != 0:

                # Find the average win ratio for decks with the present number
of unique cards.
                average_win_ratio = np.mean(list_of_win_ratios_for_decks_with_
```

```
j_unique_cards)

                # Add the average win ratio to a list of average win ratios fo
r all possible numbers of unique cards.
                average_win_ratios[j] = average_win_ratio

        # Add the present grouping of categories to a list.
        list_of_groupings.append(grouping_of_categories)

        # Add the highest of the average win ratios to a list.
        list_of_highest_average_ratios.append(average_win_ratios.max())

# Write a table of combinations of categories and corresponding highest averag
e win ratios to a CSV file.
# The table will be sorted in descending order by highest average win ratio.
combination_and_ratio_database = pd.DataFrame(list_of_highest_average_ratios,
columns=["Highest Average Ratio"], index=list_of_groupings)
combination_and_ratio_database = combination_and_ratio_database.sort_values(by
=["Highest Average Ratio"], ascending=False)
combination_and_ratio_database.to_csv("./Data_With_Ravnica_Allegiance/Category
_Combinations_and_Ratios.csv")
```

2.18. Weighting Categories in a Chosen Grouping

Return to Tasks Checklist

I wrote the below Python program to weight categories in grouping that looks promising for enjoyable and competitive gameplay. The initial weighting for a category is based on how much that category seems to contribute to the win ratio associated with the present grouping. I calculate the initial weighting of each category as the difference between the win ratio associated with the present grouping of categories and the win ratio associated with a grouping of categories without the present category. This calculation may be zero or negative, because I might choose a grouping of categories in my categories of interest database to try in MTG Arena that has a theoretical win ratio equal to or slightly less than the theoretical win ratio associated with a grouping that is my chosen grouping less the present category.

After acquiring the initial weighting of each category in the present grouping, I scale all the calculated weightings by dividing them by the smallest magnitude among weightings, and round these scaled weightings to the nearest whole number. I adjust all zero and negative weightings to 1, because I am insisting on including cards from categories that seem to dilute the potency of a theoretically more competitive deck. I interpret the scaled weighting for a category as the number of cards of that category that I want in a deck with no maximum number of cards.

After acquiring the scaled weighting / importance of each category in the present grouping, I re-scale and re-round all the scaled importances so that there sum is close to 36 (i.e., the number of nonland cards I want in a 60-card MTG deck). Again, if a re-scaled importance of a category is zero, because the importances of the other categories are so much higher, I force the re-scaled importance to 1.

My end goal is to have a sum of importances equal to 36, so that I can have a number of nonland cards equal to the importance of each category representing that category in MTG Arena deck. While the sum of importances is not equal to 36, I increase the importance of "flying", because I believe that all of my MTG Arena decks at this point need as many good flyers as they can afford, and re-scale and re-round until the sum of importances equals 36.

Please see below screenshot of "Categories_and_Importances.csv".

Category	Importance
create creature token or convert lands into creatures	12
destroy	11
draw	10
put nonland card	1
flying	2

Figure 18: Screenshot of "Categories_and_Importances.csv"

In [4]:
```
# Weighting_Categories_in_a_Chosen_Grouping.py
#
# This program creates a table of categories in a chosen grouping
# and the calculated weights or numbers of cards that should
# represent each category in a MTG-Arena deck of the chosen grouping.
#
# Created: 04/02/19 by Tom Lever
# Updated: 04/20/19 by Tom Lever
#
# Inputs: Category_Combinations_and_Ratios.csv
# Dependencies: pandas
# Outputs: Categories_and_Importances.csv


# Allow use of the pandas.read_csv method.
import pandas as pd


##############################
# Create a list of categories.
##############################

# Read the table of groupings of categories and associated highest average win
ratios into a dataframe.
categories_and_ratios = pd.read_csv("./Data_With_Ravnica_Allegiance/Category_C
ombinations_and_Ratios.csv", index_col=0, header=0)

# Select a row corresponding to a grouping to create an table of category weig
hts / importances.
desired_row = 7

# Convert the string representing the chosen grouping of categories into a lis
t of categories.
list_of_categories = categories_and_ratios.index[desired_row][1:-1].split(", "
)
list_of_categories = [category[1:-1] if category else None for category in lis
t_of_categories]

#################################################################
# Create a list of initial category weightings / importances.
#################################################################

list_of_importances = []

# For each category in the above list of categories...
for category in list_of_categories:

    # Create a string version of the above list of categories, but without the
present category.
    list_of_categories_wo_present_category = []
    for i in range(0, len(list_of_categories)):
        if list_of_categories[i] != category:
            list_of_categories_wo_present_category.append(list_of_categories[i
])
    str_of_categories_wo_present_category = str(tuple(list_of_categories_wo_pr
esent_category))
```

```
    for grouping in categories_and_ratios.index:
        if grouping == str_of_categories_wo_present_category:
            importance = categories_and_ratios.iloc[desired_row]["Highest Aver
age Ratio"] - categories_and_ratios.loc[grouping]["Highest Average Ratio"]
            list_of_importances.append(importance)

######################################################################
# Create a table of categories and initial weightings / importances.
######################################################################

categories_and_importances = pd.DataFrame(list_of_importances, columns=["Impor
tance"], index=list_of_categories)
categories_and_importances.sort_values(by="Importance", ascending=False)

print(categories_and_importances)

#####################################################################
# Scale the initial weightings / importances
# by dividing them by the smallest magnitude among importances,
# and rounding to the nearest whole number.
#
# Force all non-positive importances to 1.
#####################################################################

scaled_cats_and_imps = categories_and_importances / abs(categories_and_importa
nces["Importance"]).min()
scaled_cats_and_imps = scaled_cats_and_imps.round(0)

forced_importance = 1
for category in scaled_cats_and_imps.index:
    if scaled_cats_and_imps.loc[category]["Importance"] <= 0:
        scaled_cats_and_imps.at[category, "Importance"] = forced_importance


#####################################################################
# Re-scale the scaled importances so that there sum is close to 36.
#
# Force all importances of zero to 1.
#####################################################################

scaled_cats_and_imps = scaled_cats_and_imps / scaled_cats_and_imps["Importanc
e"].sum() * 36
scaled_cats_and_imps = scaled_cats_and_imps.round(0)

for category in scaled_cats_and_imps.index:
    if scaled_cats_and_imps.loc[category]["Importance"] == 0:
        scaled_cats_and_imps.at[category, "Importance"] = forced_importance


#################################################################
# While the sum of the scaled importances is greater than 36,
# increase importances of 1 by 1, re-scale, and re-round.
#################################################################

while scaled_cats_and_imps["Importance"].sum() != 36:
    if "flying" in scaled_cats_and_imps.index.tolist():
```

```
        scaled_cats_and_imps.loc["flying"] += 1
    else:
        scaled_cats_and_imps.loc["flying"] = 1

        scaled_cats_and_imps = scaled_cats_and_imps / scaled_cats_and_imps["Im
portance"].sum() * 36
        scaled_cats_and_imps = scaled_cats_and_imps.round(0)


#########################################################################
# Output the table of categories and scaled importances to a CSV file.
#########################################################################

scaled_cats_and_imps.to_csv("./Data_With_Ravnica_Allegiance/Categories_and_Imp
ortances.csv")
```

	Importance
boost power	0.596798
flying	0.351913
create or replace creature token| or convert in...	1.023457
destroy	0.948752
draw	1.249023

2.19. Developing Strong Deck List with Specific Mana Types, Weighted Categories, and Maximum Average Converted Mana Cost

Return to Tasks Checklist

I wrote the below Python program to develop a strong deck with mana types specified while filtering the filled-in categories database, rules-text categories specified when developing the categories of interest database and weighted according to a table of categories and importances, and a maximum average converted mana cost among nonland cards specified in this program. I start with a unrealistically high max ave CMC (i.e., 12). I vary the max ave CMC based on whether I feel that the base unconstrained deck is too "slow" / "heavy". This program will be used to play enjoyable and competitive games. Please see below screenshot of "Strong_Deck.csv".

My program requires the path to the filtered filled in categories database. My program requires the path to the categories and importances table. My program requires the path to the best cards with inventory database. My program requires the path to the condensed cards database. My program relies on the pandas Python library to read and manipulate this databases. My program relies on the random Python class to remove a card among cards with highest converted mana cost to lower average mana cost, and to increase or decrease a number of a certain randomly chosen basic land so to get closer to a total land count of 24. I rely on MTG Arena's basic land calculator to determine actual numbers of basic lands in the strong deck that I play.

Name	Average Frequency	Inventory	Desired Count	Card Type	Converted Mana Cost	Category
History of Benalia	4	2	1	Enchantment	3	create creature token or convert lands into creatures
History of Benalia	4	2	1	Enchantment	3	create creature token or convert lands into creatures
Resplendent Angel	2	1	1	Creature	3	create creature token or convert lands into creatures
March of the Multitudes	3	1	1	Instant	3	create creature token or convert lands into creatures
Karn| Scion of Urza	2	1	1	Legendary Planeswalker	4	create creature token or convert lands into creatures
Ajani| Adversary of Tyrants	2	1	1	Legendary Planeswalker	4	create creature token or convert lands into creatures
Divine Visitation	2	1	1	Enchantment	5	create creature token or convert lands into creatures
Tithe Taker	3	2	1	Creature	2	create creature token or convert lands into creatures
Tithe Taker	3	2	1	Creature	2	create creature token or convert lands into creatures
Emmara| Soul of the Accord	3	2	1	Legendary Creature	2	create creature token or convert lands into creatures
Emmara| Soul of the Accord	3	2	1	Legendary Creature	2	create creature token or convert lands into creatures
Dawn of Hope	2	4	1	Enchantment	2	create creature token or convert lands into creatures
Angel of Grace	2	1	1	Creature	5	destroy
Trapjaw Tyrant	1	1	1	Creature	5	destroy
Cleansing Nova	2	4	1	Sorcery	5	destroy
Cleansing Nova	2	4	1	Sorcery	5	destroy
Settle the Wreckage	2	1	1	Instant	4	destroy
Knight of Autumn	2	2	1	Creature	3	destroy
Knight of Autumn	2	2	1	Creature	3	destroy
Citywide Bust	2	1	1	Sorcery	3	destroy
Conclave Tribunal	3	4	1	Enchantment	4	destroy
Conclave Tribunal	3	4	1	Enchantment	4	destroy
Conclave Tribunal	3	4	1	Enchantment	4	destroy
The Immortal Sun	1	1	1	Legendary Artifact	6	draw
Dawn of Hope	2	4	1	Enchantment	2	draw
Pelakka Wurm	1	1	1	Creature	7	draw
Mentor of the Meek	1	1	1	Creature	3	draw
Camaraderie	1	3	1	Sorcery	6	draw
Fountain of Renewal	3	4	1	Artifact	1	draw
Fountain of Renewal	3	4	1	Artifact	1	draw
Fountain of Renewal	3	4	1	Artifact	1	draw
Colossal Majesty	2	2	1	Enchantment	3	draw
Colossal Majesty	2	2	1	Enchantment	3	draw
Elvish Clancaller	4	2	1	Creature	2	put nonland card
Lyra Dawnbringer	2	1	1	Legendary Creature	5	flying
Shalai| Voice of Plenty	2	2	1	Legendary Creature	4	flying
Plains	9	24	8	Basic Land	0	Basic Land
Forest	7	24	7	Basic Land	0	Basic Land
Sunpetal Grove	3	1	1	Land	0	Land
Detection Tower	1	3	1	Land	0	Land
Field of Ruin	1	2	1	Land	0	Land
Zhalfirin Void	2	1	1	Land	0	Land
Selesnya Guildgate	3	4	3	Land	0	Land
Tranquil Expanse	2	4	2	Land	0	Land

Figure 19: Screenshot of "Strong Deck.csv"

In [5]:
```
# Developing_Strong_Deck_List_with_Specific_Mana_Types_Weighted_Categories_and
_Max_Ave_CMC.py
#
# This program creates a strong deck list with mana types specified while filt
ering the filled-in categories database,
# rules-text categories specified when developing the categories-of-interest d
atabase
# and weighted when developing the categories and importances table,
# and a maximum average converted mana cost specified in this program.
#
# Created: 04/02/19 by Tom Lever
# Updated: 04/20/19 by Tom Lever
#
# Inputs: Filtered_Filled_In_Categories_Database.csv, Categories_and_Importanc
es.csv,
# Best_Cards--with_Inventory.csv, Condensed_Cards_Database.csv
# Dependences: pandas, numpy
# Outputs: Strong_Deck.csv


# Allow reading and manipulating dataframes
import pandas as pd

# Allow use of the random.choice method.
import random

# Allow creation of columns of zeros.
import numpy as np


################################################
# Create a table of names and inventories
# for all cards in one or more categories
# in the table of categories and importances.
################################################

# Read the filtered filled in categories database into a dataframe.
filtered_filled_in_categories_database = pd.read_csv("./Data_With_Ravnica_Alle
giance/Filtered_Filled_In_Categories_Database.csv", index_col=0, header=0)

# Read the categories and importances table into a dataframe.
categories_and_importances = pd.read_csv("./Data_With_Ravnica_Allegiance/Categ
ories_and_Importances.csv", index_col=0)
list_of_categories = categories_and_importances.index.tolist()

# Filter the filled in categories database into rows
# corresponding to cards in the present grouping of categories.
mask = filtered_filled_in_categories_database[list_of_categories[0]] == 1
for i in range(1, len(list_of_categories)):
    mask = mask | filtered_filled_in_categories_database[list_of_categories[i
]] == 1
cards_in_categories = filtered_filled_in_categories_database[mask]

# Create a table of card names and frequencies.
names_n_invs = pd.DataFrame(cards_in_categories["Inventory"],
                            columns=["Inventory"],
```

```
                                index=cards_in_categories.index.tolist())


##############################################################################
########
# Create a table of names, average frequencies, and inventories,
# that happens to be sorted first by rarity and second by total number of occu
rrences.
##############################################################################
########

# Enter the best cards with inventory database into a dataframe.
best_cards_with_inventory = pd.read_csv("./Data_With_Ravnica_Allegiance/Best_C
ards--with_Inventory.csv",
                                        header=0,
                                        index_col=0)

# Create a table of card names and average frequencies from the best cards dat
abase.
names_n_freqs = pd.DataFrame(best_cards_with_inventory["Average Frequency"].ro
und(decimals=0),
                             columns=["Average Frequency"],
                             index=best_cards_with_inventory.index.tolist())

# Merge the names and inventories dataframe
# for cards of appropriate mana types in the present grouping of categories
# into the names and average frequencies dataframe
# from the best-cards with inventory database,
# preserving the order of the best-cards database.
names_freqs_n_invs = names_n_freqs.merge(names_n_invs, left_index=True, right_
index=True)


##############################################################################
##
# Create a table of names, average frequencies, inventories, and desired count
s.
##############################################################################
##

# Create a column for desired counts.
desired_counts = []

# For each row in the names, frequencies, and inventories dataframe...
for i in range(0, names_freqs_n_invs.shape[0]):

    # If the inventory of the card in the present row is greater than the aver
age frequency of the card...
    if names_freqs_n_invs.iloc[i]["Inventory"] > names_freqs_n_invs.iloc[i]["A
verage Frequency"]:

        # Add the average frequency to the column of desired counts for the pr
esent card.
        desired_counts.append(names_freqs_n_invs.iloc[i]["Average Frequency"])

    # If the inventory of the card in the present row is less than the average
frequency of the card...
```

```
    else:

        # Add the inventory to the column of desired counts for the present ca
rd.
        desired_counts.append(names_freqs_n_invs.iloc[i]["Inventory"])

    # Alternative option:
    #desired_counts.append(1)

# Create a names, average frequencies, inventories, and desired counts datafra
me.
names_freqs_invs_n_counts = names_freqs_n_invs
names_freqs_invs_n_counts["Desired Count"] = desired_counts


##############################################################################
########################
# Create a table of names, average frequencies, inventories, desired counts, a
nd converted mana costs.
##############################################################################
########################

# Enter the condensed cards daabase into a dataframe.
condensed_cards_database = pd.read_csv("./Data_With_Ravnica_Allegiance/Condens
ed_Cards_Database.csv", index_col=0)

# Create a names and converted mana costs dataframe.
names_n_CMCs = pd.DataFrame(condensed_cards_database[["Card Type", "Converted
 Mana Cost"]],
                            columns=["Card Type", "Converted Mana Cost"],
                            index=condensed_cards_database.index.tolist())

# Merge the names and converted mana costs dataframe
# into the names, average frequencies, inventories, and desired counts datafra
me,
# preserving the order of the best-cards database.
names_freqs_invs_counts_n_CMCs = names_freqs_invs_n_counts.merge(names_n_CMCs,
left_index=True, right_index=True)
names_freqs_invs_counts_n_CMCs[0:5]


##############################################################################
####################################
# Copy the names, average frequencies, inventories, desired counts, and conver
ted mana costs dataframe in memory.
# The distinct dataframe in memory will be used as a checklist for reorganizin
g the cards
# corresponding to the chosen mana types and grouping of categories according
 to the importances of the categories.
##############################################################################
####################################

checklist = names_freqs_invs_counts_n_CMCs.copy()

# For each card in the checklist...
for name in checklist.index.tolist():
```

```
    # Drop all lands from the checklist.
    # Drop very specific cards from the checklist.
    if "land" in condensed_cards_database.loc[name]["Card Type"].lower():
        checklist = checklist.drop(index=name)
    if name in ["Awakened Amalgam",
                "Desecrated Tomb",
                "Dragon's Hoard",
                "Glass of the Guildpact",
                "Guild Summit",
                "Sai| Master Thopterist"]:
        checklist = checklist.drop(index=name)

# Add an "Added" column to keep track of which cards have been added
# to a list of card names that will be used as the index for
# a strong deck list, the source of strong decks with varying average mana cos
ts.
checklist["Added"] = np.zeros(checklist.shape[0])


##############################
# Create a strong-deck source.
##############################

# Create storage for a list of card names that will serve as the index for the
strong-deck source.
list_of_cards = []

# Create storage for a column of categories, one category for each card in the
strong-deck_source.
column_of_cats = []


# Create a (number of categories) x (maximum number of cards in a category) da
taframe
# with all the cards in all of the categories.
list_of_cards_in_category_for_each_category = []
for category in list_of_categories:
    list_of_cards_in_category_for_each_category.append(filtered_filled_in_cate
gories_database[filtered_filled_in_categories_database[category] == 1].index.t
olist())
categories_and_cards_in_category = pd.DataFrame(list_of_cards_in_category_for_
each_category, index=list_of_categories)


# While cards remain in the checklist...
while checklist.shape[0] > 0:

    # For each category in our present grouping of categories...
    for category in list_of_categories:

        # For each slot for a card in the present category in the strong-deck
 source that should be filled
        # before any cards representing any other categories are added to the
 strong-deck source...
        for j in range(0, int(categories_and_importances.loc[category]["Import
ance"])):
```

```
            # For each card in the checklist...
            for k in range(0, checklist.shape[0]):

                # If fewer copies of the present card have been added to the s
trong-deck source than are desired, and
                # if the present card represents the present category...
                if (checklist.iloc[k]["Added"] < checklist.iloc[k]["Desired Co
unt"]) & (checklist.iloc[k].name in categories_and_cards_in_category.loc[categ
ory].tolist()):

                    # Fill in a slot in index for the strong-deck source with
 the present card.
                    list_of_cards.append(checklist.iloc[k].name)

                    # Record the present category, which the present card is r
epresenting.
                    column_of_cats.append(category)

                    # Note in the checklist that the present card has been add
ed to the strong-deck source.
                    row = checklist.iloc[k]
                    row.at["Added"] += 1
                    checklist.iloc[k] = row

                    # Move on to the next slot in the strong-deck source.
                    break

    # Remove cards from the checklist that have had all their desired copies a
dded to the strong-deck source.
    checklist = checklist[checklist["Added"] != checklist["Desired Count"]].co
py()

# Create storage for the strong-deck source.
strong_deck_list = pd.DataFrame([], columns=names_freqs_invs_counts_n_CMCs.col
umns, index=[])

# Each card in the index for the strong-deck source...
for name in list_of_cards:

    # Copy the row from the names, frequencies, desired counts, and converted
 mana costs dataframe
    # corresponding to the present card to the strong-deck source.
    row = names_freqs_invs_counts_n_CMCs.loc[name]
    row.at["Desired Count"] = 1
    strong_deck_list = strong_deck_list.append(row)

# Add the column of categories that each card represents to the strong-deck so
urce.
strong_deck_list["Category"] = column_of_cats


##############################################################################
# Create a strong deck
# with the mana types from the filtered filled-in cards database,
# cards representing the categories in the present grouping
# in proportions specified in the table of categories and importances, and
# a specific average converted mana cost.
```

```
################################################################################

# Assume the nonland cards in the strong deck
# are the 36 best cards in the strong-deck source,
# regardless of average converted mana cost.
strong_deck = strong_deck_list.iloc[0:36]

# While the average converted mana cost among copies of the nonland cards
# is greater than some user-defined maximum...
while np.mean(strong_deck["Converted Mana Cost"]) > 3:

    # Find the maximum converted mana cost among nonland cards.
    max_CMC = strong_deck["Converted Mana Cost"].max()

    # Create a slice of the strong deck with just copies of cards with the max
imum CMC.
    most_costly_cards__rows = strong_deck[strong_deck["Converted Mana Cost"] =
= max_CMC]

    # Drop duplicates from the slice of the strong deck with copies of cards w
ith the max CMC.
    most_costly_cards__rows__dups_dropped = most_costly_cards__rows.drop_dupli
cates()

    # Create a list of the names of cards in the condensed slice.
    most_costly_cards__names__dups_dropped = most_costly_cards__rows__dups_dro
pped.index.tolist()

    # Select a card from the list of cards in the condensed slice.
    card_to_replace__name = random.choice(most_costly_cards__names__dups_dropp
ed)

    # Find the row in the condensed slice corresponding to the selected card.
    card_to_replace__row = most_costly_cards__rows__dups_dropped.loc[card_to_r
eplace__name]

    # Record the converted mana cost of the selected card,
    # which is the maximum converted mana cost.
    card_to_replace__CMC = card_to_replace__row["Converted Mana Cost"]

    # Find the category of the selected card.
    card_to_replace__category = card_to_replace__row["Category"]

    # For each row corresponding to a card in the strong deck...
    for i in range(0, strong_deck.shape[0]):

        # If the present row is matches the card to replace...
        if strong_deck.iloc[i].equals(card_to_replace__row):

            # Record the position of the row to replace in the strong deck.
            pos_of_card_to_replace_in_strong_deck = i

            # Stop searching the strong deck for the row to replace.
            break

    # Send the index of the strong deck to a list.
    strong_deck__names = strong_deck.index.tolist()
```

```
    # Remove the name of the selected card to remove from the index for the st
rong deck.
    del strong_deck__names[pos_of_card_to_replace_in_strong_deck]

    # Send the categories that cards in the strong deck represent to a list.
    strong_deck__categories = strong_deck["Category"].tolist()

    # Remove the category of the selected c ard to remove from the column of c
ategories.
    del strong_deck__categories[pos_of_card_to_replace_in_strong_deck]

    # Recreate the strong deck without the removed card.
    strong_deck = pd.DataFrame([], columns=names_freqs_invs_counts_n_CMCs.colu
mns, index=[])
    for name in strong_deck__names:
        row = names_freqs_invs_counts_n_CMCs.loc[name]
        row.at["Desired Count"] = 1
        strong_deck = strong_deck.append(row)
    strong_deck["Category"] = strong_deck__categories

    # Keep track of whether a replacement card with a lesser mana cost than th
at of the removed card
    # has been added to the strong deck.
    was_card_added = False

    # For each card back in the strong-deck list...
    for i in range(0, strong_deck_list.shape[0]):

        # If the converted mana cost of the present card is less than the CMC
 of the card that was replaced...
        if strong_deck_list.iloc[i]["Converted Mana Cost"] < card_to_replace__
CMC:

            # If the present card represents the category that the removed car
d represented...
            if strong_deck_list.iloc[i]["Category"] == card_to_replace__catego
ry:

                # If the number of copies of the present card in the new stron
g deck
                # is less than the desired count for copies of the present car
d...
                if strong_deck[strong_deck.index == strong_deck_list.iloc[i].n
ame].shape[0] < strong_deck_list.iloc[i]["Desired Count"]:

                    # Add the row corresponding to the replacement card from t
he strong-deck source
                    # to the strong deck.
                    strong_deck = strong_deck.append(strong_deck_list.iloc[i])

                    # Note that a card was added to the strong deck to replace
the removed card.
                    was_card_added = True

                    # Stop looking for a replacement card.
                    break
```

```
    # If no card was found in the strong-deck list that met all the criteria a
bove...
    if not was_card_added:

        # For each card in the strong-deck source...
        for i in range(0, strong_deck_list.shape[0]):

            # If the CMC of the present card is less than the CMC of the card
 that was replaced...
            if strong_deck_list.iloc[i]["Converted Mana Cost"] < card_to_repla
ce__CMC:

                # If the present card in the strong-deck source represents the
"flying" category...
                if strong_deck_list.iloc[i]["Category"] == "flying":

                    # If the number of copies of the present card in the new s
trong deck
                    # is less than the desired count for copies of the present
card...
                    if strong_deck[strong_deck.index == strong_deck_list.iloc[
i].name].shape[0] < strong_deck_list.iloc[i]["Desired Count"]:

                        # Add the row corresponding to the replacement card fr
om the strong-deck source
                        # to the strong deck.
                        strong_deck = strong_deck.append(strong_deck_list.iloc
[i])

                        # Note that a card was added to the strong deck to rep
lace the removed card.
                        was_card_added = True

                        # Stop looking for a replacement card.
                        break


##########################################
# Add lands at the end of the strong deck.
##########################################

# Create a table of names and inventories for all cards in the filtered filled
-in categories database.
names_n_invs = pd.DataFrame(filtered_filled_in_categories_database["Inventory"
].tolist(),
                            columns=["Inventory"],
                            index=filtered_filled_in_categories_database.ind
ex.tolist())

# Create a dataframe of names, rarities, total number of occurrences, and aver
age frequencies
# for all cards in the best cards with inventory database.
names_rarities_occurs_n_freqs = pd.DataFrame([],
                                             columns=["Rarity", "Total Number
 of Occurrences", "Average Frequency"],
                                             index=best_cards_with_inventory.i
```

```
ndex.tolist())
names_rarities_occurs_n_freqs["Rarity"] = best_cards_with_inventory["Rarity"]
names_rarities_occurs_n_freqs["Total Number of Occurrences"] = best_cards_with
_inventory["Total Number of Occurrences"]
names_rarities_occurs_n_freqs["Average Frequency"] = best_cards_with_inventory
["Average Frequency"]

# Merge the names and inventories table into the names, rarities, occurrences,
and frequencies dataframe,
# preserving the order of the best-cards database.
names_rarities_occurs_freqs_n_invs = names_rarities_occurs_n_freqs.merge(names
_n_invs, left_index=True, right_index=True)

# Create a names and card types table from the condensed cards database.
names_n_card_types = pd.DataFrame(condensed_cards_database["Card Type"].tolist
(),
                                  columns=["Card Type"],
                                  index=condensed_cards_database.index.tolist
())

# Create a dataframe of information on lands with appropriate mana types.
filtered_best_lands = names_rarities_occurs_freqs_n_invs.merge(names_n_card_ty
pes,
                                                              left_index=True
,
                                                              right_index=Tru
e)
filtered_best_lands = filtered_best_lands[filtered_best_lands["Card Type"].str
.lower().str.contains("land")]
filtered_best_lands = filtered_best_lands.round(0)

# Drop lands that I don't like from the dataframe of lands.
for name in filtered_best_lands.index.tolist():
    if name in ["Plaza of Harmony", "Reliquary Tower", "Rupture Spire", "Gatew
ay Plaza", "Unknown Shores"]:
        filtered_best_lands = filtered_best_lands.drop(index=name).round(0)

# Create a list of desired counts for the lands based on average frequencies a
nd inventories.
desired_counts = []
for i in range(0, filtered_best_lands.shape[0]):
    if filtered_best_lands.iloc[i]["Inventory"] > filtered_best_lands.iloc[i][
"Average Frequency"]:
        desired_counts.append(filtered_best_lands.iloc[i]["Average Frequency"
])
    else:
        desired_counts.append(filtered_best_lands.iloc[i]["Inventory"])

# Add the desired count list as a column to the dataframe of information on la
nds.
filtered_best_lands["Desired Count"] = desired_counts

# Add a column of zeros indicating converted mana costs of zero for lands to t
he dataframe of information on lands.
filtered_best_lands["Converted Mana Cost"] = np.zeros(filtered_best_lands.shap
e[0])
```

```
# Add a column indicating that lands fall into their own category of "Land".
filtered_best_lands["Category"] = filtered_best_lands["Card Type"]
filtered_best_lands = filtered_best_lands[["Average Frequency", "Inventory",
"Desired Count", "Card Type", "Converted Mana Cost", "Category"]]

# Play with the numbers of basic lands to get the total number of lands to 24.
while filtered_best_lands["Desired Count"].sum() < 24:
    basic_land_to_increase = random.choice(["Forest", "Island", "Mountain", "P
lains", "Swamp"])
    if basic_land_to_increase in filtered_best_lands.index.tolist():
        filtered_best_lands.at[basic_land_to_increase, "Desired Count"] += 1
while filtered_best_lands["Desired Count"].sum() > 24:
    basic_land_to_increase = random.choice(["Forest", "Island", "Mountain", "P
lains", "Swamp"])
    if basic_land_to_increase in filtered_best_lands.index.tolist():
        filtered_best_lands.at[basic_land_to_increase, "Desired Count"] -= 1

# Add 24 appropriate lands with category "Land" to the strong deck.
strong_deck = strong_deck.append(filtered_best_lands)


################################################################################
##################
# Output the strong deck to a CSV file.
# Output the total number of copies of cards in the deck to this notebook.
# Output the average converted mana cost among nonland cards in the strong dec
k to this notebook.
################################################################################
##################

# Write the strong deck to a CSV file.
strong_deck.to_csv("./Data_With_Ravnica_Allegiance/Strong_Deck.csv")

print("The total number of copies of cards, including lands, in \"Strong_Deck.
csv\" is " + "%.f" % strong_deck["Desired Count"].sum() + ".")
print("The average converted mana cost among copies of nonland cards in \"Stro
ng_Deck.csv\" is " + "%.3f" % np.mean(strong_deck["Converted Mana Cost"][stron
g_deck["Converted Mana Cost"] > 0]) + ".")
```

```
The total number of copies of cards, including lands, in "Strong_Deck.csv" is
60.
The average converted mana cost among copies of nonland cards in "Strong_Dec
k.csv" is 3.000.
```

7. Findings

7.1. An FP deck

In "Ranked" mode in MTG Arena, at Gold Tier 3, a (C / F / P / CF / CP / FP / CFP), a ("create creature token or convert lands into creatures", "destroy", "draw", "put nonland card", "flying") deck, with an unconstrained maximum average mana cost among nonland cards and an actual cost of 3.444 (which felt a little heavy):

1. Lost against an unblockable / power up / draw / counter / hexproof / boost toughness; Mist-Cloaked Herald / Curious Obsession / Pteramander; I deck.
2. Won against a menace / deathtouch / first strike; Immolation Shaman / Goblin Chainwhirler; MS deck.
3. Won against an Orzhov Enforcer; PS deck.

8. Limitations

There are two major limitations associated with the win / loss ratios for the many decks that ultimately impact the accuracy of the win / loss ratios associated with groupings of categories. First, while I obviously ignored decks without win / loss ratios, I did not ignore decks with win / loss ratios of 0 or 100. Given my play results and experience, it seems unlikely that so many decks would have long-term average win / loss ratios of 0 or 100. I think eliminating these decks too would increase the accuracy and actually provide more spread to the win / loss ratios associated with groupings of categories. However, I preferred keeping them to increase the number of cards in the best-cards with inventory database and to spread the total numbers of occurrences of these cards.

Closely related to the above limitation, I was not able to easily scrape together information on how many game results were included in the total win and total loss numbers for each deck. There is an implicit assumption that each deck was played the same number of times and an infinite number of times. I believe such total win and total loss number information can be found by interpreting graphs on each of MTGArena.pro’s deck pages. However, it did seem that many of the decks were played for similar amounts of time, which might suggest that they were played a similar number of times.

9. Conclusion

By writing and using software to answer my research question of, basically, “What is a strong-deck list meeting the five specified criteria in the research question section?”, I assembled a strong-deck list that turned out to win two out of the three games that I played with it. I would say that this method of finding strong deck lists is a “win”.

10. Acknowledgements and References

I am grateful to:

1. EdX.org and the University of California: San Diego for offering a free "Python for Data Science" course geared toward graduate students, which got me started with Python, database manipulation, machine-learning, and natural-language processing.
2. Wizards of the Coast for offering a web-based MTG cards database and a gameplay environment that are easy and enjoyable to use.
3. MTGArena.pro for offering an extensive community-supported decks database.
4. All the contributors to StackOverflow.com for providing solutions and workarounds to conception and implementation difficulties I ran into in writing the Jupyter Notebook.
5. The community-oriented (I assume) authors of all the Python modules (including web scraping, database manipulation, and machine-learning modules) that were essential to my work.
6. The authors of the approachable and practical "Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit", which provides me food for thought in how to analyze my rules text database more effectively and begin to study natural language processing and artificial intelligence.