In [45]:
```
### Written by vapicuno, May 17, 2023
### Viability ranking analysis of 2023 RBY OU metagame
### VR post https://www.smogon.com/forums/threads/rby-ou-viability-rankings.3685861/page-5
### Based on the method described at
### https://www.smogon.com/forums/threads/quantitative-tiering-in-viability-rankings.3649801/

### README
### Run all cells in sequential order
### Read the comments before each cell
### Usually, where it is expected that an edit has to be made, you will see the following tags
### EDIT THE FOLLOWING
### END EDIT

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms
import matplotlib.patches as patches
import matplotlib.cm as cm
import matplotlib.colors as colors
%matplotlib inline
import pandas as pd
import itertools
from timeit import default_timer as timer
from scipy import stats
from scipy import cluster
import pandas as pd
import copy
```

In [46]:
```
### Initialization

## EDIT THE FOLLOWING

# csv input with header=voters and row labels=Pokemon
# Remember to remove any columns that don't form part of the original data, li
ke "Average"
fnIn = '2023_RBY_rankings_outliers_in.csv' # this update's data input
fnOut = '2023_RBY_rankings_outliers_out.csv' # this script will output this up
date's outlier-compensated data here
fnPrevIn = '2022_RBY_rankings_outliers_in.csv' # last update's data input
fnPrevOut = '2022_RBY_rankings_outliers_out.csv' # this script will output las
t update's outlier-compensated data here
cutoff = 19 # cut off analysis at this number of Pokemon
dropFromVR = 4 # if the number of people who voted for this Pokemon is less th
an this, then exclude from final VR

gen = 'RBY OU' # Tier name
year = '2023' # Update indicator, can be year or season
lastyear = '2022' # Last update indicator, can be year or season

## END EDIT

rank = pd.read_csv(fnIn,header=0,index_col=0)
rankval = rank.values
N = np.shape(rankval)
rankFracThreshold = 1-dropFromVR/N[1] # throw out Pokemon that are not ranked
by over this fraction of voters
```

In [47]:
```
### Outlier compensation
# This shifts outliers to the boundary of a reference range
# If an outlier seems too far out, reduce the outlier boundary variable "limi
t"
# Usually no change is needed
# Look at the outlier-compensated ranks by running the next code block and ope
ning the output csv in Excel
rankvalNoOutliers = copy.deepcopy(rankval)
Q1 = np.quantile(rankval,0.16,axis=1) # set reference range
Q3 = np.quantile(rankval,0.84,axis=1) # set reference range
IQR = Q3 - Q1
limit = 1 # set outlier boundary
for ii in range(0,rankval.shape[0]):
    lb = Q1[ii] - limit*IQR[ii]
    ub = Q3[ii] + limit*IQR[ii]
    for jj in range(0,rankval.shape[1]):
        if rankval[ii,jj] == np.nan:
            rankvalNoOutliers[ii,jj] = np.nan
        elif rankval[ii,jj] < lb:
            rankvalNoOutliers[ii,jj] = lb
        elif rankval[ii,jj] > ub:
            rankvalNoOutliers[ii,jj] = ub
```

In [48]:
```
### FILE OUTPUT 1
### Outputs outlier compensated ranks to csv
rankNoOutliers = pd.DataFrame(rankvalNoOutliers)
rankNoOutliers.to_csv(fnOut, header=False, index=False)
```

Tiering Analysis

In [49]:
```
### Sort data by outlier-removed ranks
rankval = rank.values
rankavg = np.nanmean(rankvalNoOutliers,axis=1)
rankstd = np.nanstd(rankvalNoOutliers,axis=1)
orderNoOutliers = np.argsort(rankavg)
rankavg = rankavg[orderNoOutliers]
rankstd = rankstd[orderNoOutliers]
monList = rank.index.values
monList = monList[orderNoOutliers]
rankval = rankval[orderNoOutliers,:]
# Ranking Difference
rankMeanSubtracted = rankval - np.repeat(np.array([rankavg]).transpose(),rankv
al.shape[1],axis=1)
```

In [50]:
```
### TEXT OUTPUT 1
### Final VR order
for n in range(0,N[0]):
    print(str(n+1).zfill(2) + ' ' + monList[n])
```

```
01 Tauros
02 Snorlax
03 Chansey
04 Starmie
05 Exeggutor
06 Alakazam
07 Rhydon
08 Zapdos
09 Gengar
10 Cloyster
11 Jynx
12 Slowbro
13 Jolteon
14 Lapras
15 Persian
16 Articuno
17 Victreebel
18 Moltres
19 Dragonite
20 Porygon
21 Golem
22 Rapidash
23 Kingler
24 Sandslash
25 Hypno
26 Ninetales
27 Kabutops
28 Arbok
29 Kangaskahn
30 Poliwrath
31 Venusaur
32 Gyarados
33 Raichu
34 Raticate
35 Pinsir
36 Machamp
37 Flareon
38 Nidoqueen
39 Clefable
40 Omastar
41 Lickitung
```

In [51]:
```
### TEXT OUTPUT 2
### Final VR order after removing lesser-ranked Pokemon
orderCount = 0
monCount = 1
orderReduced = []
for n in range(0,N[0]):
    orderCount += 1
    if sum(np.isnan(rankval[n,:])) > rankFracThreshold*N[1]:
        continue
    print(str(monCount).zfill(2) + ' ' + monList[n])
    orderReduced += [orderCount-1]
    monCount += 1
```

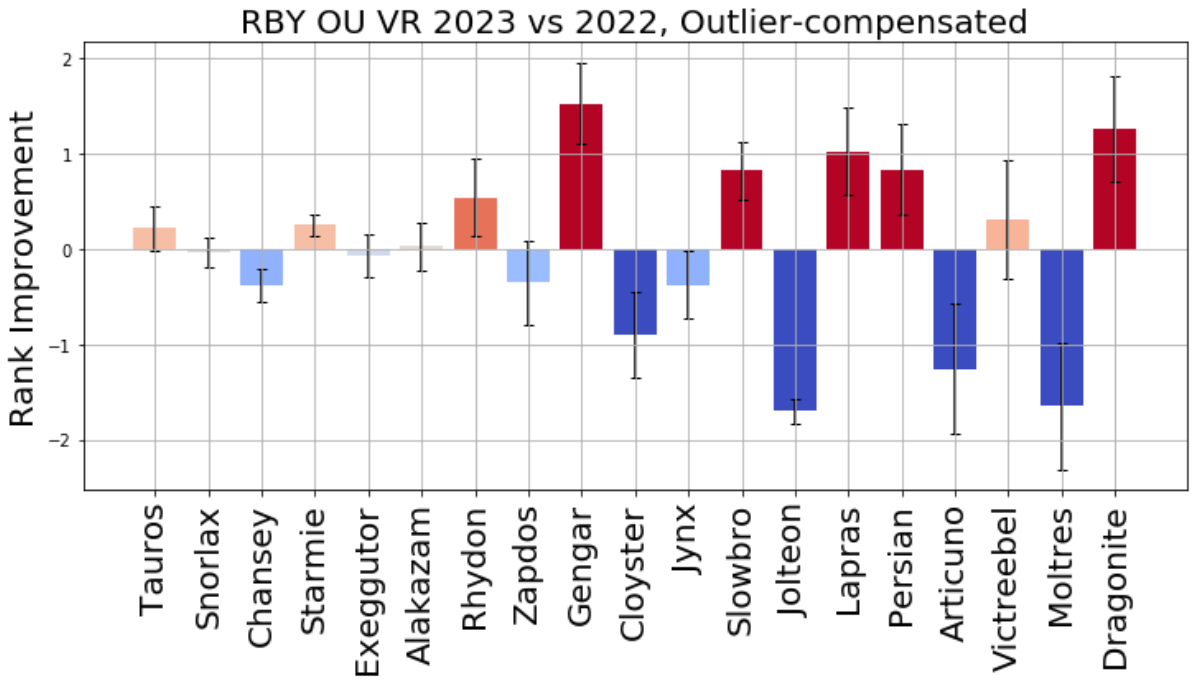
```
01 Tauros
02 Snorlax
03 Chansey
04 Starmie
05 Exeggutor
06 Alakazam
07 Rhydon
08 Zapdos
09 Gengar
10 Cloyster
11 Jynx
12 Slowbro
13 Jolteon
14 Lapras
15 Persian
16 Articuno
17 Victreebel
18 Moltres
19 Dragonite
20 Porygon
21 Golem
22 Kingler
23 Sandslash
24 Hypno
25 Kabutops
26 Kangaskahn
27 Poliwrath
28 Venusaur
29 Raichu
30 Raticate
31 Machamp
32 Clefable
33 Omastar
```

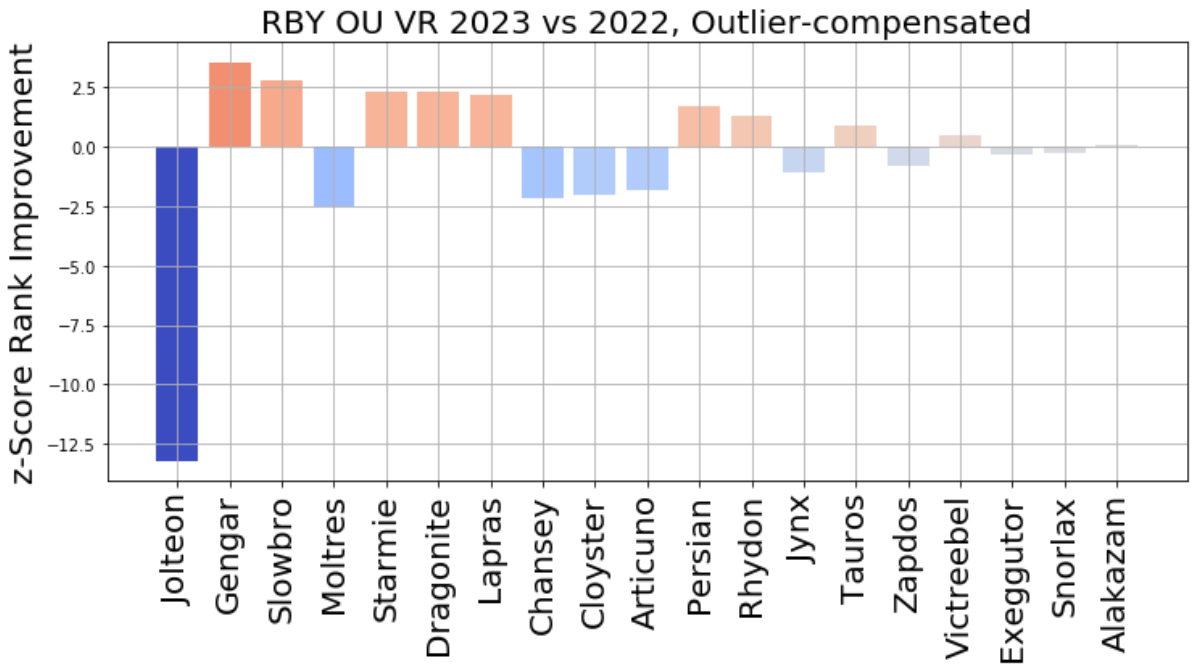
In [52]:
```
### Analyze previous rankings
rankOld = pd.read_csv(fnPrevIn,header=0,index_col=0)
rankvalOld = rankOld.values
Nold = np.shape(rankvalOld)

rankNoOutliersOld = pd.DataFrame(rankvalNoOutliers)
rankNoOutliersOld.to_csv(fnPrevOut, header=False, index=False)

rankvalNoOutliersOld = copy.deepcopy(rankvalOld)
Q1old = np.quantile(rankvalOld,0.16,axis=1)
Q3old = np.quantile(rankvalOld,0.84,axis=1)
IQRold = Q3old - Q1old
limitOld = 1
for ii in range(0,rankvalOld.shape[0]):
    lb = Q1old[ii] - limitOld*IQRold[ii]
    ub = Q3old[ii] + limitOld*IQRold[ii]
    for jj in range(0,rankvalOld.shape[1]):
        if rankvalOld[ii,jj] == np.nan:
            rankvalNoOutliersOld[ii,jj] = np.nan
        elif rankvalOld[ii,jj] < lb:
            rankvalNoOutliersOld[ii,jj] = lb
        elif rankvalOld[ii,jj] > ub:
            rankvalNoOutliersOld[ii,jj] = ub

rankavgOld = np.nanmean(rankvalNoOutliersOld,axis=1)
rankstdOld = np.nanstd(rankvalNoOutliersOld,axis=1)
orderNoOutliersOld = np.argsort(rankavgOld)
rankavgOld = rankavgOld[orderNoOutliersOld]
rankstdOld = rankstdOld[orderNoOutliersOld]
monListOld = rankOld.index.values
monListOld = monListOld[orderNoOutliersOld]

indexOldIntoNew = [list(monListOld).index(mon) for mon in monList[:cutoff]]
rankOldIntoNew = rankavgOld[indexOldIntoNew]
rankDiff = rankavg[:cutoff] - rankOldIntoNew
```

```
C:\Users\lambj\anaconda3\lib\site-packages\ipykernel_launcher.py:25: RuntimeW
arning: Mean of empty slice
C:\Users\lambj\anaconda3\lib\site-packages\numpy\lib\nanfunctions.py:1667: Ru
ntimeWarning: Degrees of freedom <= 0 for slice.
  keepdims=keepdims)
```

In [53]:
```
### GRAPHIC OUTPUTS 1 AND 2
### Compare current and previous rankings
# The first graphic shows rank improvements sorted by VR rank
# The second graphic shows z-score rank improvements sorted by z-score
# The z-score identifies the changes that are most statistically significant
# It weighs the change in ranking by the spread of the ranking

seismic = cm.get_cmap('coolwarm', 32)
rankDiffStd = np.sqrt(rankstdOld[:cutoff]**2/Nold[1]+rankstd[:cutoff]**2/N[1])
zChange = -rankDiff / (rankDiffStd+1e-7)

fig = plt.figure(figsize=(12,5))
ax = plt.axes()
ax.grid(zorder=0)
ax.bar(list(monList[:cutoff]),list(-rankDiff),zorder=1,color=seismic(0.5-rankD
iff/max(rankDiff)),yerr=rankDiffStd,ecolor='k',capsize=3)
plt.xticks(rotation=90,fontsize=min(580/cutoff,20))
plt.ylabel('Rank Improvement',fontsize=20)
plt.title(gen + ' VR ' + year + ' vs ' + lastyear + ', Outlier-compensated',fo
ntsize=20)

orderChange = sorted(range(len(monList[:cutoff])),reverse=True,key=lambda x:n
p.abs(zChange[x]))
fig = plt.figure(figsize=(12,5))
ax = plt.axes()
ax.grid(zorder=0)
ax.bar(list(monList[:cutoff][orderChange]),list(zChange[orderChange]),zorder=
1,color=seismic(0.5+zChange[orderChange]/max(np.abs(zChange))))
plt.xticks(rotation=90,fontsize=min(580/cutoff,20))
plt.ylabel('z-Score Rank Improvement',fontsize=20)
plt.title(gen + ' VR ' + year + ' vs ' + lastyear + ', Outlier-compensated',fo
ntsize=20)
```

Out[53]: Text(0.5, 1.0, 'RBY OU VR 2023 vs 2022, Outlier-compensated')



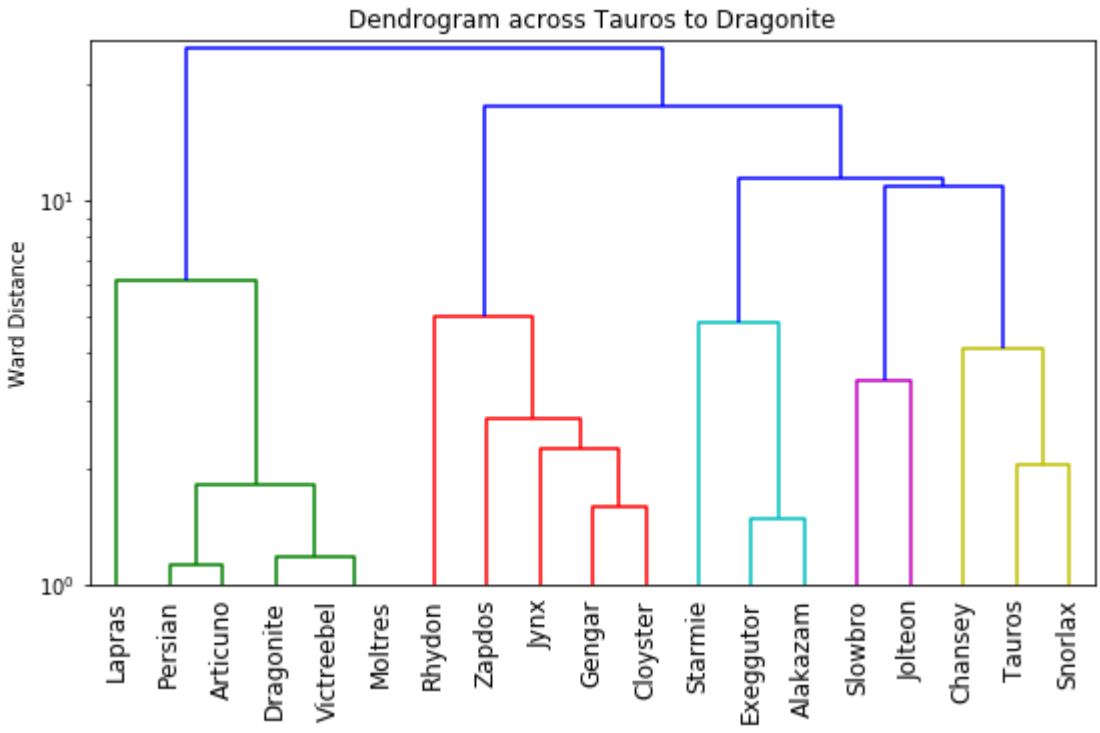


In [54]:
```
### Handle non-ranked Pokemon by substituting with rank 999 and eliminating co
rrelation in case of equal ranks
maxDist = 4
dissimMons = np.zeros((cutoff,cutoff))
def compareArr(arr1,arr2):
    assert len(arr1) == len(arr2)
    N = len(arr1)
    arr1cp = copy.deepcopy(arr1)
    arr2cp = copy.deepcopy(arr2)
    logicalArr = np.zeros(N)
    arr1cp[np.isnan(arr1cp)] = 999
    arr2cp[np.isnan(arr2cp)] = 999
    for n in range(N):
        if arr1cp[n] == arr2cp[n]:
            logicalArr[n] = 0.5
        else:
            logicalArr[n] = arr1cp[n] > arr2cp[n]
    return logicalArr

### Perform rank rate comparison across Pokemon pairs and perform logit transf
orm
for m1 in range(cutoff):
    for m2 in range(cutoff):
        if m1 == m2:
            continue
        rate = sum(compareArr(rankval[m1,:],rankval[m2,:])) / N[1]
        dissimMons[m1,m2] = min(abs(np.log(rate/(1-rate))),maxDist)
```

```
C:\Users\lambj\anaconda3\lib\site-packages\ipykernel_launcher.py:25: RuntimeW
arning: divide by zero encountered in log
C:\Users\lambj\anaconda3\lib\site-packages\ipykernel_launcher.py:25: RuntimeW
arning: divide by zero encountered in double_scalars
```

In [55]:
```
### GRAPHIC OUTPUT 3
### Performs hierarchical clustering with Ward linkage and plots a dendrogram
# Hierarchical clustering helps identify tiers that are not as easy to discern
by eye
# This usually does a good job at identifying the tiers
# Sometimes it may be erroneous and need modifying in the codeblock below
# Set tierThreshold to where you want tiers to be defined
# Tiers will appear as groups of different colors
### EDIT THE FOLLOWING
tierThreshold = 7.5 # set this to the tiering threshold
### END EDIT
Hmons = cluster.hierarchy.ward(dissimMons)
fig = plt.figure(figsize=(9,5))
ax = plt.gca()
ax.set_yscale('log')
ax.set_ylim([1,50])
ax.set_ylabel('Ward Distance')
Dmons = cluster.hierarchy.dendrogram(Hmons,labels=[monList[i] for i in range(c
utoff)],leaf_rotation=90,color_threshold=tierThreshold,leaf_font_size=12)
clustersMons = cluster.hierarchy.fcluster(Hmons, tierThreshold, criterion='dis
tance')
plt.title('Dendrogram across ' + monList[0] + ' to ' + monList[cutoff-1]);
```

```
C:\Users\lambj\anaconda3\lib\site-packages\scipy\cluster\hierarchy.py:2834: U
serWarning: Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
  ax.set_ylim([0, dvw])
```



In [56]:
```
### TIER DETERMINATION

### MODIFIYING DENDROGRAM-ASSIGNED TIERS
# Sometimes, the dendrogram assigns tiers wrongly, which can be identified upo
n
# examining the dissimilarity matrix three codeblocks below
# Uncomment the code below to reassign misassigned Pokemon to their correct ti
ers.
# For example, the code below reassigns Charizard to be in the same tier as Es
peon
# and reassigns Smeargle to the same tier as Dragonite.

# clustersMons[list(monList).index('Charizard')] = clustersMons[list(monList).
index('Espeon')]
# clustersMons[list(monList).index('Smeargle')] = clustersMons[list(monList).i
ndex('Dragonite')]

### Obtain clusters
clusterDividers = [0] + [clustersMons[i+1]-clustersMons[i] != 0 for i in range
(len(clustersMons)-1)] + [1]
clusterIndices = np.insert(np.nonzero(clusterDividers)[0],0,0)

### ADD TIER INDICES
# Sometimes, you may want to artificially split a tier in two.
# Also, the dendrogram only creates tiers up till the user-defined cutoff
# Lower tiers will have to be self-defined by the visual method described in
# https://www.smogon.com/forums/threads/quantitative-tiering-in-viability-rank
ings.3649801/
# This code inserts tier boundaries in the positions indicated below
# In this specific instance, tier boundaries are inserted after Pokemon 26 and
31 (Kangaskhan and Dodrio)
# len(monList) is there to indicate that the end of the lowest tier is the end
of the list of ranked Pokemon

### EDIT THE FOLLOWING
clusterIndicesMod = np.append(clusterIndices,[27,len(monList)])
### END EDIT
clusterIndicesMod.sort()

### ADD TIER NAMES
### EDIT THE FOLLOWING
tierNames = ['S','A','B1','B2','C','D','E']
### END EDIT
```

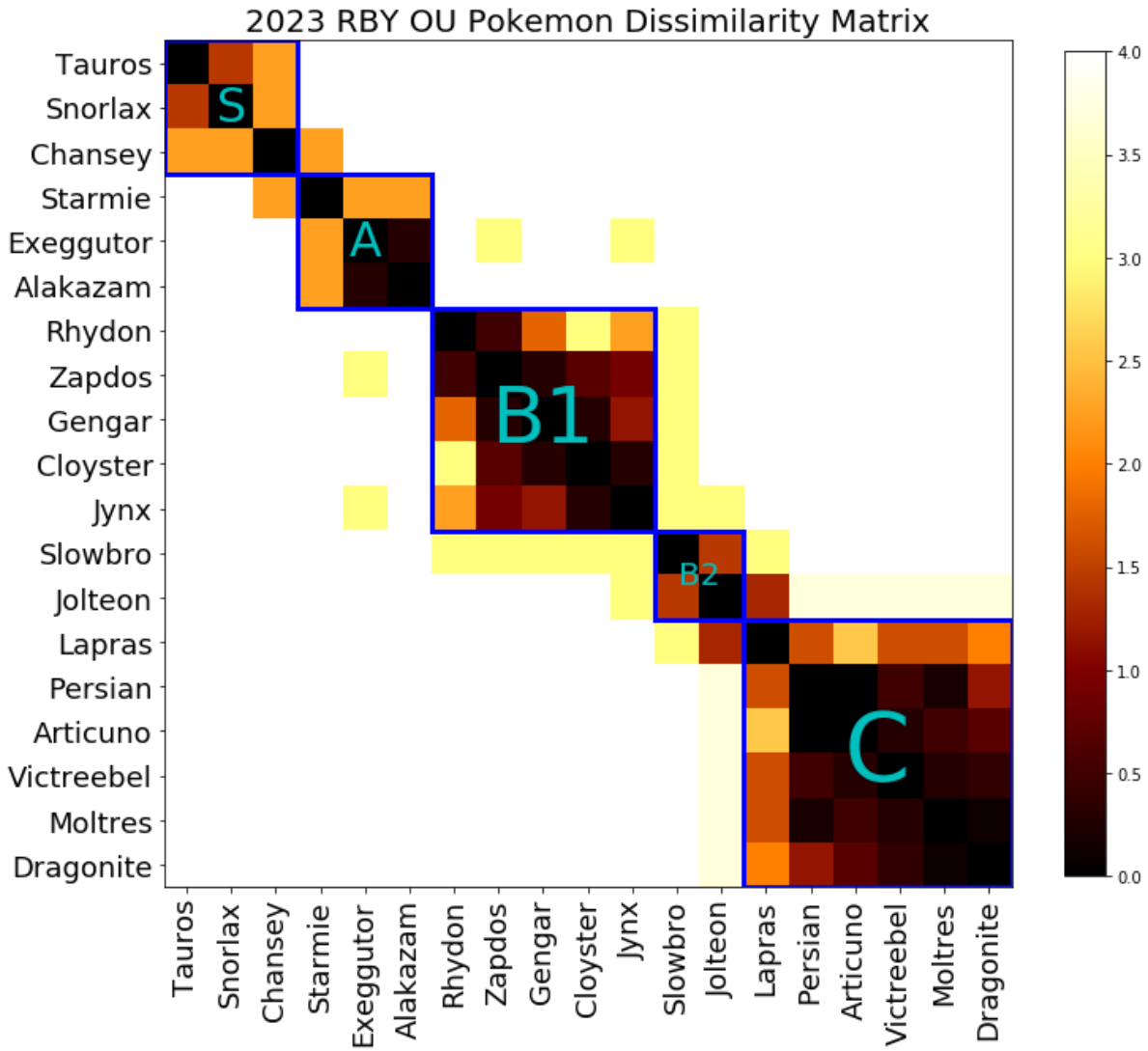
In [57]:
```
### CHECK TIERS
# If the dendrogram-assigned tiers have been modified, this checks the modific
ation is correct
# The tier numbers are listed in the order of the dendrogram
clustersMons
```

Out[57]:
```
array([5, 5, 5, 3, 3, 3, 2, 2, 2, 2, 2, 4, 4, 1, 1, 1, 1, 1, 1],
      dtype=int32)
```

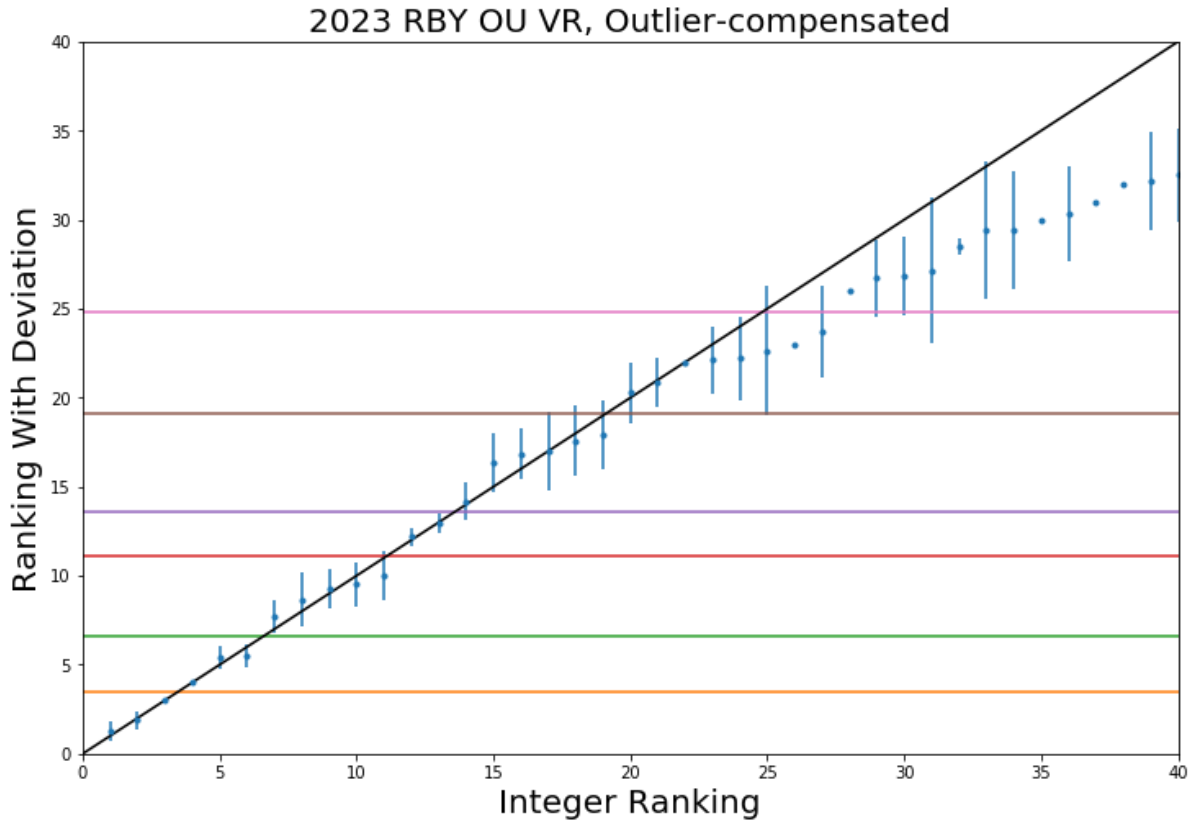
In [58]:
```
### TEXT OUTPUT 3
### Print VR tier writeup to be used in Smogon forums
textVR = ''
for n in range(len(clusterIndicesMod)-1):
    textVR += tierNames[n] + ': '
    for m in range(clusterIndicesMod[n],clusterIndicesMod[n+1]):
        if sum(np.isnan(rankval[m,:])) > rankFracThreshold*N[1]:
            continue
        textVR += ':' + monList[m] + ':'
    textVR += '\n'
print(textVR)
```

```
S: :Tauros::Snorlax::Chansey:
A: :Starmie::Exeggutor::Alakazam:
B1: :Rhydon::Zapdos::Gengar::Cloyster::Jynx:
B2: :Slowbro::Jolteon:
C: :Lapras::Persian::Articuno::Victreebel::Moltres::Dragonite:
D: :Porygon::Golem::Kingler::Sandslash::Hypno::Kabutops:
E: :Kangaskahn::Poliwrath::Venusaur::Raichu::Raticate::Machamp::Clefable::Oma
star:
```

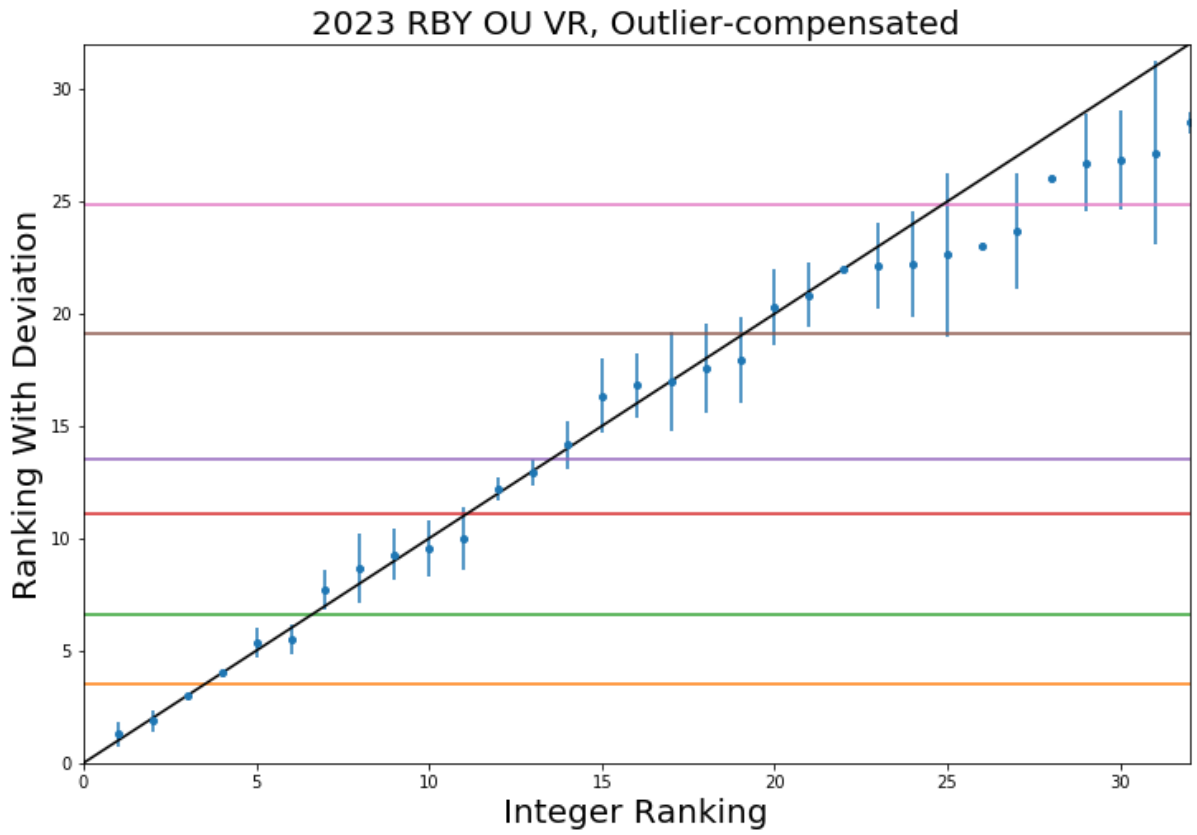
In [59]:
```
### GRAPHIC OUTPUT 4
### Calculate dissimilarity matrix from rank rates across pairs of Pokemon.
# Pokemon that are similar in rank on average will appear as darker squares.
# Dendrogram tiers are displayed as boxes and should mostly match dark square
s.
# If any Pokemon is clearly misplaced, it can be modified three codeblocks abo
ve.
fig = plt.figure(figsize=(12,12))
ax = plt.gca()
h = ax.imshow(dissimMons,interpolation='none',cmap='afmhot',vmin=0,vmax=4)
for c in range(len(clusterIndicesMod)-1):
    ax.plot([clusterIndicesMod[c]-0.5,clusterIndicesMod[c+1]-0.5],[clusterIndi
cesMod[c]-0.5,clusterIndicesMod[c]-0.5],color='b',linewidth=3)
    ax.plot([clusterIndicesMod[c]-0.5,clusterIndicesMod[c+1]-0.5],[clusterIndi
cesMod[c+1]-0.5,clusterIndicesMod[c+1]-0.5],color='b',linewidth=3)
    ax.plot([clusterIndicesMod[c]-0.5,clusterIndicesMod[c]-0.5],[clusterIndice
sMod[c]-0.5,clusterIndicesMod[c+1]-0.5],color='b',linewidth=3)
    ax.plot([clusterIndicesMod[c+1]-0.5,clusterIndicesMod[c+1]-0.5],[clusterIn
dicesMod[c]-0.5,clusterIndicesMod[c+1]-0.5],color='b',linewidth=3)
fig.colorbar(h,ax=ax,shrink=0.8)
# ax.set_title('Dissimilarity Across S/A/B' + ' Camps\n Red = ' + namesA[camps
[0][0]] + ' Camp Favors Y; ' + namesA[camps[1][0]] + ' Camp Favors X\n Blue =
' + namesA[camps[1][0]] + ' Camp Favors Y; ' + namesA[camps[0][0]] + ' Camp Fa
vors X',fontsize=18)
ax.set_xticks(range(0,dissimMons.shape[1]))
ax.set_yticks(range(0,dissimMons.shape[0]))
ax.set_xticklabels([monList[i] for i in range(cutoff)],fontsize=18)
ax.set_yticklabels([monList[i] for i in range(cutoff)],fontsize=18)
ax.set_xlim([-1/2,cutoff-1/2])
ax.set_ylim([cutoff-1/2,-1/2])
ax.set_title(year + ' ' + gen + ' Pokemon Dissimilarity Matrix',fontsize=20)
plt.setp(ax.get_xticklabels(), rotation=90, ha="right",va="center",rotation_mo
de="anchor");
for ii in range(len(clusterIndicesMod)-1):
    pos = (clusterIndicesMod[ii]+clusterIndicesMod[ii+1]-1)/2
    if pos > cutoff:
        continue
    tierFontSize = (clusterIndicesMod[ii+1]-clusterIndicesMod[ii])*10
    text = ax.text(pos, pos, tierNames[ii],ha="center",va="center",color="c",f
ontsize=tierFontSize)
```


2023 RBY OU Pokemon Dissimilarity Matrix
Tauros
Snorlax
Chansey
Starmie
Exeggutor
Alakazam
Rhydon
Zapdos
Gengar
Cloyster
Jynx
Slowbro
Jolteon
Lapras
Persian
Articuno
Victreebel
Moltres
Dragonite
S
A
B1
B2
C
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

In [60]:
```
### GRAPHIC OUTPUT 5
### Draw VR linear plot as per
### https://www.smogon.com/forums/threads/quantitative-tiering-in-viability-rankings.3649801/

monsLim = 40 # decide on a reasonable limit
# monsLim = len(monList)
fig = plt.figure(figsize=(12,8))
ax = plt.axes()
plt.errorbar(np.arange(1,N[0]+1),rankavg,yerr=rankstd,marker='.',linestyle='none')
for c in clusterIndicesMod[1:-1]:
    plt.plot([0,monsLim],[(rankavg[c-1]+rankavg[c])/2]*2)
plt.plot([0,monsLim],[0,monsLim],color='k')
plt.xlim([0,monsLim])
plt.ylim([0,monsLim])
plt.xlabel('Integer Ranking',fontsize=20)
plt.ylabel('Ranking With Deviation',fontsize=20)
plt.title(year + ' ' + gen + ' VR, Outlier-compensated',fontsize=20);
```



In [61]:
```
### GRAPHIC OUTPUT 6
### Draw zoomed in VR linear plot
# This is the plot I like to add sprites to for visualization

monsLimZoom = 32 # zoom to this number
fig = plt.figure(figsize=(12,8))
ax = plt.axes()
plt.errorbar(np.arange(1,N[0]+1),rankavg,yerr=rankstd,marker='.',linestyle='no
ne',markersize=8)
for c in clusterIndicesMod[1:-1]:
    plt.plot([0,monsLim],[(rankavg[c-1]+rankavg[c])/2]*2)
plt.plot([0,monsLim],[0,monsLim],color='k')
plt.xlim([0,monsLimZoom])
plt.ylim([0,monsLimZoom])
plt.xlabel('Integer Ranking',fontsize=20)
plt.ylabel('Ranking With Deviation',fontsize=20)
plt.title(year + ' ' + gen + ' VR, Outlier-compensated',fontsize=20);
```



Camps Analysis

In [62]:
```
### Again substitute non-ranked Pokemon with 999
rankvalSub = copy.deepcopy(rankval)
for m in np.arange(0,cutoff):
    for n in np.arange(0,N[1]):
        if np.isnan(rankval[m,n]):
            rankvalSub[m,n] = 999
```

In [67]:
```
### Calculate ranking correlations for dissimilarity matrix across voters
C = len(clusterIndicesMod) - 1
totalCountArr = np.zeros((N[1],N[1],C,C))
cumulativeCountArr = np.zeros((N[1],N[1],C,C))
corrArr = np.zeros((N[1],N[1],C,C))
tArr = np.zeros((N[1],N[1],C,C))
for n1 in np.arange(0,N[1]):
    start = timer();
    for n2 in np.arange(0,N[1]):
        if n1 > n2:
            corrArr[n1,n2,:,:] = corrArr[n2,n1,:,:]
            totalCountArr[n1,n2,:,:] = totalCountArr[n2,n1,:,:]
        elif n1 == n2:
            continue
        else:
            for c1 in range(C):
                for c2 in range(c1+1):
                    for m1 in range(clusterIndicesMod[c1],clusterIndicesMod[c1
+1]):
                        for m2 in range(clusterIndicesMod[c2],clusterIndicesMo
d[c2+1]):
                            if m2 < m1:
                                corr = np.sign(rankvalSub[m1,n1]-rankvalSub[m
2,n1])*np.sign(rankvalSub[m1,n2]-rankvalSub[m2,n2])
                                corrArr[n1,n2,c1,c2] += corr
                                if corr != 0:
                                    totalCountArr[n1,n2,c1,c2] += 1

for c1 in range(C):
    for c2 in range(c1+1):
        tArr[:,:,c1,c2] = np.sum(corrArr[:,:,c2:c1,c2:c1],axis=(2,3))
        cumulativeCountArr[:,:,c1,c2] = np.sum(totalCountArr[:,:,c2:c1,c2:c1],
axis=(2,3))

tArr = tArr / cumulativeCountArr
for n1 in range(N[1]):
    tArr[n1,n1,:,:] = 1
```

```
C:\Users\lambj\anaconda3\lib\site-packages\ipykernel_launcher.py:21: RuntimeW
arning: invalid value encountered in sign
C:\Users\lambj\anaconda3\lib\site-packages\ipykernel_launcher.py:31: RuntimeW
arning: invalid value encountered in true_divide
```

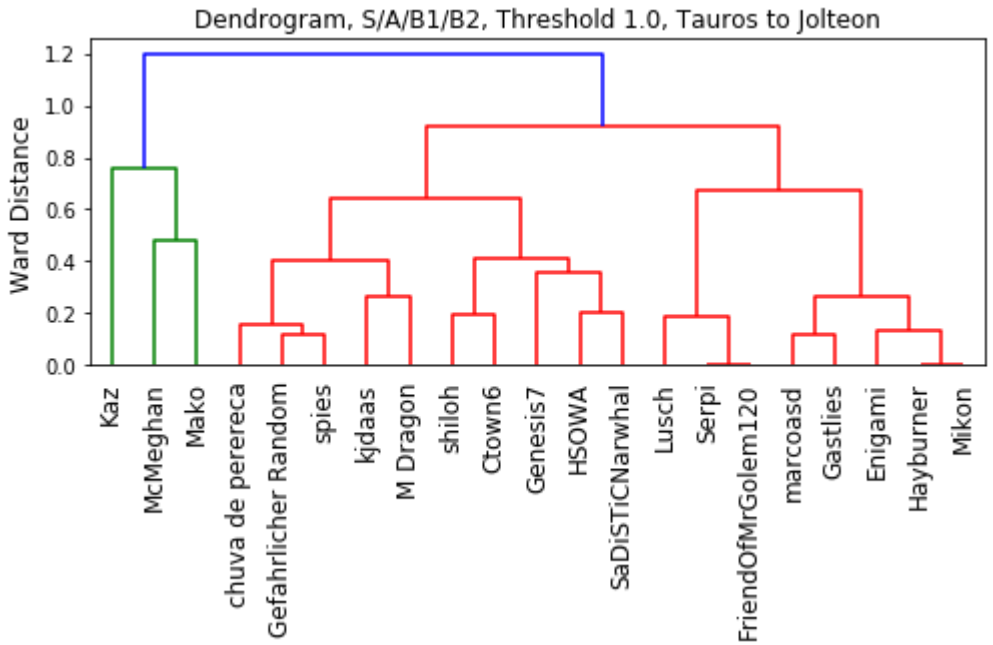
Camp Determination

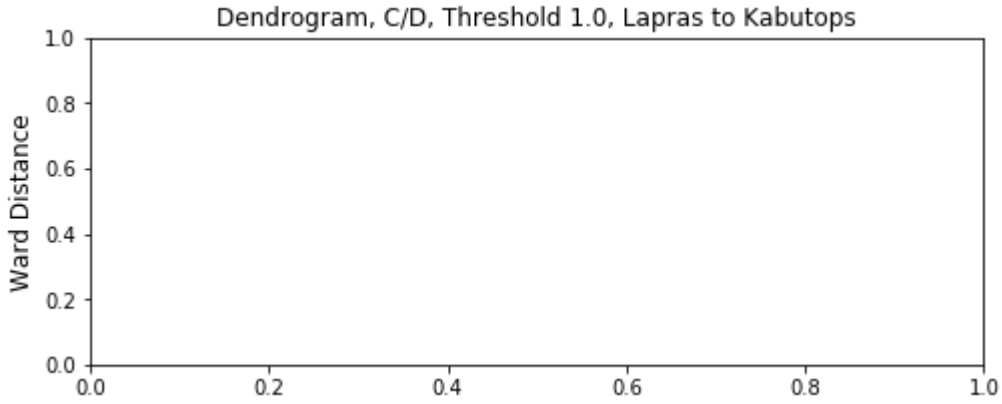
This section is about finding divides in opinions, and is the part that requires the most effort. You have to do the work of identifying the groups of tiers where these divisions can be found, through the "segments" and "thresholdSegments" variable. Change the groups of tiers, perhaps following some rule of thumb like S-to-B for a start, to identify these divides.

A good indication that you have found a strong divide is the presence of two or three long branches in the dendrogram, and huge z-scores in the relative rankings chart. The former means the colored "leaves" should be very low compared to the blue "branches" connecting them. An example of an ideal divide is the first dendrogram below, where the Nails-TC camp's "leaves" are all connected at height of 0.6, whereas the branch connecting them and the other camps has height 1.75 which is much larger than 0.6. A less-than-ideal divide is that of the Enigami-ErPeris and Genesis7-spies camps because their branches connect at a relatively low height of 1.1.

In [68]:
```
### GRAPHIC OUTPUT 7
### Find dendrogram for interested tiers
# The groups of tiers to investigate are written in a list in the variable called "segments"
# Each entry of segments captures the span from the tier represented by the first to second index
# The indices correspond to tier divisions defined by the tierNames and clusterIndicesMod

# Example: the tiers defined above were 'S','A','B1','B2','C1','C2','D1','D2','E'
# The "segments" variable is configured to find divisions across the tiers 0-4, 4-6, and 0-6,
# which translates to S-B2, C1-C2, S-C2
# (Python's notation ignores the last number specified, so 0-4 means 0,1,2,3)

# The variable "thresholdSegments" determines the height at which camps are delineated.
# Example: the first dendrogram is cut off at a height of 1, leaving three camps.
# If instead the first dendrogram was cut off at 1.5, two camps would be produced
# The second dendrogram is cut off at a height of 3.5, leaving 2 camps.
# The third dendrogram is cut off at a height of 0.6, leaving 3 camps.

### EDIT THE FOLLOWING
segments = [[0,4],[4,6],[0,6]]
thresholdSegments = [1,1,0.6]
### END EDIT

names = list(rank.columns)
Hlist = []
Dlist = []

for i in range(len(segments)):
    pair = segments[i]
    fig = plt.figure(figsize=(8,3))
    ax = plt.axes()
    ax.set_ylabel('Ward Distance',fontsize=12)
    title = 'Dendrogram, ' + tierNames[pair[0]]
    for n in range(pair[0]+1,pair[1]):
        title += '/' + tierNames[n]
    title += ', Threshold ' + '{:.1f}'.format(thresholdSegments[i])
    title += ', ' + monList[clusterIndicesMod[pair[0]]] + ' to ' + monList[clusterIndicesMod[pair[1]]-1]

    ax.set_title(title,fontsize=12)
    Hlist += [cluster.hierarchy.ward(1-tArr[:,:,pair[1],pair[0]])]
    Dlist += [cluster.hierarchy.dendrogram(Hlist[-1],labels=names,color_threshold=thresholdSegments[i],leaf_rotation=90,leaf_font_size=12)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-68-135b007682c3> in <module>
     37
     38     ax.set_title(title,fontsize=12)
---> 39     Hlist += [cluster.hierarchy.ward(1-tArr[:,:,pair[1],pair[0]])]
     40     Dlist += [cluster.hierarchy.dendrogram(Hlist[-1],labels=names,col
or_threshold=thresholdSegments[i],leaf_rotation=90,leaf_font_size=12)]

~\anaconda3\lib\site-packages\scipy\cluster\hierarchy.py in ward(y)
    828
    829     """
--> 830     return linkage(y, method='ward', metric='euclidean')
    831
    832

~\anaconda3\lib\site-packages\scipy\cluster\hierarchy.py in linkage(y, metho
d, metric, optimal_ordering)
   1059
   1060     if not np.all(np.isfinite(y)):
-> 1061         raise ValueError("The condensed distance matrix must contain
only "
   1062                          "finite values.")
   1063

ValueError: The condensed distance matrix must contain only finite values.
```





Camp Statistics

For each dendrogram (tier group) explored, the following graphs are produced

1. Relative rankings of all camps
    - Shows both Pokemon inside and outside the tier groups explored. Those outside the tier group are shaded gray.
2. Similarity matrix of all voters grouped by camps
    - Compares the aggregate rankings only within the tier group
3. For each camp, the following graphs are produced
    A. z-score of relative rank, ordered by z-score
        - **This is the most important graph.** It highlights the significant differences in opinion in a visually pleasing manner

    B. z-score of relative rank, ordered by ranking
        - Sometimes, the graph above includes too many fringe Pokemon at the top. This still displays the statistical significance, but by the conventional ranking.

    C. Dissimilarity matrix
        - This is used to find **specific information about pairs of Pokemon, or find general trends** that cannot conventionally be seen in the z-score plots. The number and darkness on each square correlates with how much one camp's opinions on the said pair of Pokemon are inverted from the rest of the voters.
        - Example of specific pair: The Nails camp in S-B2 has a 100% inversion of Jolteon with Zapdos, indicating that everyone in this camp thinks of Jolteon as the superior Electric-type.
        - Example of general trend: The Troller camp in S-C2 mostly has similar opinions about which tiers Pokemon belong in, but have a lot of inversions within the tiers themselves, with Alakazam > Exeggutor, Cloyster/Jynx/Jolteon > Zapdos/Rhydon, and Moltres/Dragonite/Golem/Porygon > Lapras/Victreebel/Persian

In [ ]:
```
### GRAPHIC OUTPUT 8
### Generates all statistical plots based on tier spans defined above in list
"segments"
# Graphs are not produced for camps with less than ignoreCampsLessThan number
of voters

### EDIT THE FOLLOWING
ignoreCampsLessThan = 2;
### END EDIT

for n in range(len(segments)):
    pair = segments[n]
    order = Dlist[n]['leaves']
    rankMeanSubtractedCutoff = rankMeanSubtracted[:cutoff,order]
    namesSegment = [names[i] for i in order]
    # marker = itertools.cycle
(('v','^','<','>','*','d','o','p','P','h','H','X','D'))
    marker = itertools.cycle(('v','^','<','>','*','d','X'))
    clustersSegmentUnordered = cluster.hierarchy.fcluster(Hlist[n], thresholdS
egments[n], criterion='distance')
    clustersSegment = clustersSegmentUnordered[order]
    clustersSegmentDividers = [0] + [clustersSegment[i+1]-clustersSegment[i] !
= 0 for i in range(len(clustersSegment)-1)] + [1]
    clustersSegmentIndices = np.insert(np.nonzero(clustersSegmentDividers)[0],
0,0)
    segmentLen = np.diff(clustersSegmentIndices)

    ### Plot Relative Ranks
    cArr = ['r'] + ['g'] + ['b'] + ['m'] + ['c']
    cArr.insert(np.argmax(segmentLen),'y')
    color = itertools.cycle(tuple(cArr))
    delta = 0.2
    fig = plt.figure(figsize=(12,5))
    ax = plt.axes()
    ax.grid(zorder=0)
    offset = lambda p: transforms.ScaledTranslation(p/72.,0, plt.gcf().dpi_sca
le_trans)
    rankMeanSubtractedSegmentCutoff = []
    rankStdSubtractedSegmentCutoff = []
    monListCutoff = list(monList[:cutoff])
    for p in range(0,len(segmentLen)):
        offsetNum = 5*((p-len(clustersSegmentIndices)+2)/(len(clustersSegmentI
ndices)-2)+1/2) + 0
        rankMeanSubtractedSegment = rankMeanSubtractedCutoff[:,clustersSegment
Indices[p]:clustersSegmentIndices[p+1]]
        trans = plt.gca().transData
        # ax.scatter(list(monList[:Dcut]),list(-np.mean(rankMeanSubtractedADca
mp,axis=1)),err=list(np.std(rankMeanSubtractedADcamp,axis=1)),zorder=1,label=n
amesAD[p],marker = next(marker),s=200,alpha=0.5,color=cArrAD[p])
        rankMeanSubtractedSegmentCutoff += [-np.nanmean(rankMeanSubtractedSegm
ent,axis=1)]
        rankStdSubtractedSegmentCutoff += [np.nanstd(rankMeanSubtractedSegmen
t,axis=1)/np.sqrt(segmentLen[p]-1)]
        if segmentLen[p] < ignoreCampsLessThan:
            continue
        ax.errorbar(monListCutoff,list(rankMeanSubtractedSegmentCutoff[p]),lis
```

```
t(rankStdSubtractedSegmentCutoff[p]),zorder=1,label=namesSegment[clustersSegme
ntIndices[p]],
                    linestyle='None',marker=next(marker),markersize=10,alpha=
0.5,color=next(color),transform=trans+offset(offsetNum))

    ax.set_ylim(ax.get_ylim())
    ax.set_xlim(ax.get_xlim())
    ax.plot([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.5],[-
99,99],color='k')
    ax.fill([-99,-99,clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]
-0.5],[-99,99,99,-99],'k',alpha=0.1)
    ax.plot([clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.5],[-
99,99],color='k')
    ax.fill([clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.5,cut
off*2,cutoff*2],[-99,99,99,-99],'k',alpha=0.1)
    plt.xticks(rotation=90,size=18)
    ax.legend(loc='upper center',bbox_to_anchor=(1.1,1),ncol=1)
    plt.ylabel('Relative Rank In Favor',fontsize=20)
    title = year + ' ' + gen + ' VR Relative Statistics: ' + tierNames[pair
[0]]
    for m in range(pair[0]+1,pair[1]):
        title += '/' + tierNames[m]
    plt.title(title,fontsize=20)

    ### Plot Ranking Correlations
    tClustered = np.array([[tArr[i,j,pair[1],pair[0]] for j in order] for i in
order])
    for ii in range(0,tClustered.shape[0]):
        tClustered[ii,ii] = np.nan
    fig = plt.figure(figsize=(12,12))
    ax = plt.gca()
    h = ax.imshow(tClustered,interpolation='none',cmap='viridis')
    fig.colorbar(h,ax=ax,shrink=0.8)
    title = 'Ranking Correlation'
    title = tierNames[pair[0]]
    for m in range(pair[0]+1,pair[1]):
        title += '/' + tierNames[m]
    title += ', ' + monList[clusterIndicesMod[pair[0]]] + ' to ' + monList[clu
sterIndicesMod[pair[1]]-1]

    ax.set_title(title,fontsize=18)
    ax.set_xticks(range(0,tClustered.shape[1]))
    ax.set_yticks(range(0,tClustered.shape[0]))
    ax.set_xticklabels([names[i] for i in order],fontsize=18)
    ax.set_yticklabels([names[i] for i in order],fontsize=18)
    plt.setp(ax.get_xticklabels(), rotation=90, ha="right",va="center",rotatio
n_mode="anchor");
    for ii in range(0,tClustered.shape[0]):
        for jj in range(0,tClustered.shape[1]):
            if ii != jj:
                text = ax.text(ii, jj, int(100*tClustered[ii, jj]),
                               ha="center", va="center", color="w",fontsize=1
2)

    ### Plot Sorted Differences by Z Score
    rankMeanSubtractedCutoffOtherSegments = []
    rankStdSubtractedCutoffOtherSegments = []
```

```
    rankStdDifference = []
    orderSegment = []
    seismic = cm.get_cmap('coolwarm', 32)
    dataNormalizer = colors.Normalize()
    for p in range(len(segmentLen)):
        rankSubtractedCutoffOtherSegments = rankMeanSubtractedCutoff[:,np.arra
y(np.concatenate((range(clustersSegmentIndices[p]),range(clustersSegmentIndice
s[p+1],len(order)))),dtype=int)]
        rankMeanSubtractedCutoffOtherSegments += [-np.nanmean(rankSubtractedCu
toffOtherSegments,axis=1)]
        rankStdSubtractedCutoffOtherSegments += [np.nanstd(rankSubtractedCutof
fOtherSegments,axis=1)/np.sqrt(len(order)-segmentLen[p]-1)]
        rankStdDifference += [np.sqrt(rankStdSubtractedSegmentCutoff[p]**2 + r
ankStdSubtractedCutoffOtherSegments[p]**2)]
        zSegment = (rankMeanSubtractedSegmentCutoff[p] - rankMeanSubtractedCut
offOtherSegments[p]) / (rankStdDifference[p]+1e-7)
        zSegment[np.isnan(zSegment)] = 0

        orderSegment += [sorted(range(len(monListCutoff)),reverse=True,key=lam
bda x:np.abs(zSegment[x]))]
        if segmentLen[p] < ignoreCampsLessThan:
            continue
        # order by z score
        fig = plt.figure(figsize=(12,5))
        ax = plt.axes()
        ax.grid(zorder=0)
        offset = lambda p: transforms.ScaledTranslation(p/72.,0, plt.gcf().dpi
_scale_trans)
        limit = min(max(abs(zSegment[abs(zSegment)<20]))*1.05,5)
        ax.bar(range(len(monListCutoff)),zSegment[orderSegment[p]],zorder=1,co
lor=seismic(zSegment[orderSegment[p]]/(2)+0.5))
        ax.set_xticks(range(len(monListCutoff)))
        ax.set_xticklabels([monListCutoff[i] for i in orderSegment[p]],fontsiz
e=360/cutoff)
        plt.xticks(rotation=90,size=16)
        ax.set_ylabel('Z Score of Relative Rank in Favor',fontsize=15)
        title = tierNames[pair[0]]
        for m in range(pair[0]+1,pair[1]):
            title += '/' + tierNames[m]
        title += ', Threshold ' + '{:.1f}'.format(thresholdSegments[i])
        title += ', ' + namesSegment[clustersSegmentIndices[p]] + ' camp'
        title += ', ' + str(segmentLen[p]) + '/' + str(len(order)) + ' voters'
        ax.set_title(title,fontsize=18)
        ax.set_ylim([-limit,limit])

        # order by rank
        fig = plt.figure(figsize=(12,5))
        ax = plt.axes()
        ax.grid(zorder=0)
        offset = lambda p: transforms.ScaledTranslation(p/72.,0, plt.gcf().dpi
_scale_trans)
        limit = min(max(abs(zSegment[abs(zSegment)<20]))*1.05,5)
        ax.bar(range(len(monListCutoff)),zSegment,zorder=1,color=seismic(zSegm
ent/(2)+0.5))
        ax.set_xticks(range(len(monListCutoff)))
        ax.set_xticklabels(monListCutoff,fontsize=360/cutoff)
        plt.xticks(rotation=90,size=16)
```

```
        ax.set_ylabel('Z Score of Relative Rank in Favor',fontsize=15)
        title = tierNames[pair[0]]
        for m in range(pair[0]+1,pair[1]):
            title += '/' + tierNames[m]
        title += ', Threshold ' + '{:.1f}'.format(thresholdSegments[i])
        title += ', ' + namesSegment[clustersSegmentIndices[p]] + ' camp'
        title += ', ' + str(segmentLen[p]) + '/' + str(len(order)) + ' voters'
        ax.set_xlim(ax.get_xlim())
        ax.plot([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.
5],[-99,99],color='k')
        ax.fill([-99,-99,clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair
[0]]-0.5],[-99,99,99,-99],'k',alpha=0.1)
        ax.plot([clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.
5],[-99,99],color='k')
        ax.fill([clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.
5,cutoff*2,cutoff*2],[-99,99,99,-99],'k',alpha=0.1)
        ax.set_title(title,fontsize=18)
        ax.set_ylim([-limit,limit])


        ### Plot Dissimilarity Matrix
        dissimSegment = np.zeros((cutoff,cutoff))
        totalCountSegment = np.zeros((cutoff,cutoff))
        for m1 in range(cutoff):
            totalCountSegment[m1,m1] = 1
        for m1 in range(cutoff):
            for m2 in range(cutoff):
                if m1 == m2:
                    continue

                for n1 in range(clustersSegmentIndices[p],clustersSegmentIndic
es[p+1]):
                    for n2 in np.array(np.concatenate((range(clustersSegmentIn
dices[p]),range(clustersSegmentIndices[p+1],len(order)))),dtype=int):
                        sign1 = np.sign(rankvalSub[m1,order[n1]]-rankvalSub[m
2,order[n1]])
                        sign2 = np.sign(rankvalSub[m1,order[n2]]-rankvalSub[m
2,order[n2]])
                        corr = sign1 * sign2
                        if corr != 0:
                            dissimSegment[m1,m2] += - (1 - corr)/2 * sign1
                            totalCountSegment[m1,m2] += 1
        dissimSegment = dissimSegment / totalCountSegment
        for m1 in range(cutoff):
            dissimSegment[m1,m1] = 0
        fig = plt.figure(figsize=(12,12))
        ax = plt.gca()
        h = ax.imshow(dissimSegment,interpolation='none',cmap='seismic',vmin=-
1,vmax=1)
        ax.plot([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[1]]-0.
5],[clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.5],color='k')
        ax.plot([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[1]]-0.
5],[clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.5],color='k')
        ax.plot([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.
5],[clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[1]]-0.5],color='k')
        ax.plot([clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.
5],[clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[1]]-0.5],color='k')
```

```
        ax.fill([-0.5,-0.5,clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pa
ir[0]]-0.5],[-0.5,cutoff-0.5,cutoff-0.5,-0.5],'k',alpha=0.1)
        ax.fill([clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.
5,cutoff-0.5,cutoff-0.5],[-0.5,cutoff-0.5,cutoff-0.5,-0.5],'k',alpha=0.1)
        ax.fill([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.
5,clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.5],[clusterIndic
esMod[pair[1]]-0.5,cutoff-0.5,cutoff-0.5,clusterIndicesMod[pair[1]]-0.5,],'k',
alpha=0.1)
        ax.fill([clusterIndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.
5,clusterIndicesMod[pair[1]]-0.5,clusterIndicesMod[pair[1]]-0.5],[-0.5,cluster
IndicesMod[pair[0]]-0.5,clusterIndicesMod[pair[0]]-0.5,-0.5],'k',alpha=0.1)

        fig.colorbar(h,ax=ax,shrink=0.8)
        title = 'Dissimilarity '
        title += tierNames[pair[0]]
        for m in range(pair[0]+1,pair[1]):
            title += '/' + tierNames[m]
        title += ', Threshold ' + '{:.1f}'.format(thresholdSegments[i])
        title += ', ' + str(segmentLen[p]) + '/' + str(len(order)) + ' voters
in '
        title += namesSegment[clustersSegmentIndices[p]] + ' Camp'
        title += ' \n Red = ' + namesSegment[clustersSegmentIndices[p]] + ' Ca
mp Favors Y; ' + ' Remainder Favors X\n Blue = ' + ' Remainder Favors Y; ' + n
amesSegment[clustersSegmentIndices[p]] + ' Camp Favors X'
        ax.set_title(title,fontsize=18)
        ax.set_xticks(range(0,dissimSegment.shape[1]))
        ax.set_yticks(range(0,dissimSegment.shape[0]))
        ax.set_xticklabels([monList[i] for i in range(cutoff)],fontsize=min(52
0/cutoff,20))
        ax.set_yticklabels([monList[i] for i in range(cutoff)],fontsize=min(52
0/cutoff,20))
        plt.setp(ax.get_xticklabels(), rotation=90, ha="right",va="center",rot
ation_mode="anchor");
        for ii in range(0,dissimSegment.shape[0]):
            for jj in range(0,dissimSegment.shape[1]):
                if dissimSegment[ii,jj] != 0:
                    text = ax.text(jj, ii, int(100*dissimSegment[ii, jj]),
                                   ha="center", va="center", color="w",fontsiz
e=360/cutoff)
```

Display of Individual Rankings

In [ ]:
```
### GRAPHIC OUTPUT 9
### Individual Relative Ranks

# Define range for clustering
refRange = -1
# Order voters by camps
pair = segments[refRange]
order = Dlist[refRange]['leaves']
namesInd = [names[i] for i in order]

## Option 1 plots relative rank wrt mean rank
rankFinalSubtractedOrder = rankMeanSubtracted[:,order]

## Option 2 plots relative rank wrt integer rank
# rankFinalSubtracted = rankval - np.repeat(np.array([np.arange(1,N[0]+1)]).tr
anspose(),rankval.shape[1],axis=1)
# rankFinalSubtractedOrder = rankFinalSubtracted[:,order]

# Eliminate underranked Pokemon
rankFinalSubtractedOrder = rankFinalSubtractedOrder[orderReduced,:]
Nr = np.shape(rankFinalSubtractedOrder)
monListReduced = monList[orderReduced]

fig = plt.figure(figsize=(Nr[1]/3,Nr[0]/3))
ax = plt.gca()
h = ax.imshow(-rankFinalSubtractedOrder,interpolation='none',cmap='seismic',vm
in=-10,vmax=10)

fig.colorbar(h,ax=ax,shrink=0.8)
title = 'Individual Relative Rankings\n Sorted By Camps in '
title += tierNames[pair[0]]
for m in range(pair[0]+1,pair[1]):
    title += '/' + tierNames[m]
title += '\nPokemon with <' + str(int(100*(1-rankFracThreshold))) + '% votes a
re removed'
ax.set_title(title,fontsize=18)
ax.set_xticks(range(0,Nr[1]))
ax.set_yticks(range(0,Nr[0]))

ax.get_xaxis().set_visible(False)
ax1 = ax.twiny()
ax1.xaxis.tick_top()
ax2 = ax.twiny()
ax2.xaxis.tick_bottom()

ax.get_yaxis().set_visible(False)
ax3 = ax.twinx()
ax3.yaxis.tick_left()
ax4 = ax.twinx()
ax4.yaxis.tick_right()

ax1.set_xticks(np.append(np.arange(0.5,Nr[1]),Nr[1]))
ax2.set_xticks(np.append(np.arange(0.5,Nr[1]),Nr[1]))
ax3.set_yticks(np.append(Nr[0],np.arange(Nr[0]-0.5,0,-1)))
ax4.set_yticks(np.append(Nr[0],np.arange(Nr[0]-0.5,0,-1)))
ax1.set_xticklabels([namesInd[i] for i in range(Nr[1])],fontsize=360/cutoff)
```

```
ax2.set_xticklabels([namesInd[i] for i in range(Nr[1])],fontsize=360/cutoff)
ax3.set_yticklabels(['']+[monListReduced[i] for i in range(Nr[0])],fontsize=36
0/cutoff)
ax4.set_yticklabels(['']+[monListReduced[i] for i in range(Nr[0])],fontsize=36
0/cutoff)
plt.setp(ax1.get_xticklabels(), rotation=90, ha="left",va="center",rotation_mo
de="anchor");
plt.setp(ax2.get_xticklabels(), rotation=90, ha="right",va="center",rotation_m
ode="anchor");
for ii in range(0,Nr[0]):
    for jj in range(0,Nr[1]):
        if not np.isnan(-rankFinalSubtractedOrder[ii,jj]):
            text = ax.text(jj, ii, int(np.round(-rankFinalSubtractedOrder[ii,j
j])),
                           ha="center", va="center", color="w",fontsize=360/cu
toff)

ax.set_aspect('auto')
for n in np.arange(1,Nr[1]/5):
    ax.plot([n*5-0.5,n*5-0.5],[-0.5,ax.get_ylim()[0]],color='k')

tempBool = np.zeros(N[0])
tempBool[clusterIndicesMod[clusterIndicesMod!=N[0]]] = 1
tempBool = tempBool[orderReduced]
clusterIndicesModReduced = np.nonzero(tempBool)
clusterIndicesModReduced = np.append(clusterIndicesModReduced,Nr[0])

for n in clusterIndicesModReduced:
    ax.plot([-0.5,ax.get_xlim()[1]],[n-0.5,n-0.5],color='c')
plt.delaxes(fig.axes[1])
```

In [ ]: