JackD-HW6-FinalProject

February 14, 2024

1 The Importance of Preprocessing - Predicting Intelligence in Dog Breeds

1.0.1 Jack DeGesero

1.0.2 In my project, I will create a machine learning model which will predict what level of intelligence a dog breed is based on different characteristics. The learning techniques used are clustering (for preprocessing) and classification (for prediction). The goal of this project is to see the evolution of prediction accuracy before and after multiple preprocessing techniques. This data was originally scraped from https://dogtime.com/ and uploaded to kaggle (https://www.kaggle.com/datasets/yonkotoshiro/dogs-breeds/?select=dogs_cleaned.csv). The dataset used has 392 rows and 37 columns before preprocessing.

1.1 Preprocessing

[1]:
```
#import packages

import numpy as py #series'
import pandas as pd #dataframes
import re #regular expressions
import numpy as np

import seaborn as sns #graphing
import matplotlib.pyplot as plt #graphing
import plotly.express as px #graphing

from sklearn.cluster import KMeans #clustering
from sklearn.preprocessing import LabelEncoder #categorical to numeric for␣
 ↪classification input
from sklearn import tree #tree object
from sklearn import model_selection #paritioning test & training data
from sklearn import metrics #find other metrics
```

[2]:
```
df = pd.read_csv("dogs.csv")
```

[3]:
```
df
```

[3]:
```
                      Unnamed: 0   Dog Breed Group  \
0                         Afador  Mixed Breed Dogs
1                    Affenhuahua  Mixed Breed Dogs
2                  Affenpinscher    Companion Dogs
3                   Afghan Hound        Hound Dogs
4               Airedale Terrier      Terrier Dogs
..                           …               …
387  Wirehaired Pointing Griffon     Sporting Dogs
388               Xoloitzcuintli    Companion Dogs
389                Yakutian Laika      Working Dogs
390                     Yorkipoo       Hybrid Dogs
391            Yorkshire Terrier    Companion Dogs

                                               Height           Weight  \
0                                     20 to 29 inches  50 to 75 pounds
1                                      6 to 12 inches   4 to 12 pounds
2                  9 to 11 inches tall at the shoulder    7 to 9 pounds
3                 24 to 26 inches tall at the shoulder  50 to 60 pounds
4                 21 to 23 inches tall at the shoulder  40 to 65 pounds
..                                                 …              …
387               20 to 24 inches tall at the shoulder  50 to 60 pounds
388  1 foot, 6 inches to 1 foot, 11 inches tall at …  10 to 50 pounds
389                                   20 to 23 inches  40 to 55 pounds
390                7 to 15 inches tall at the shoulder   3 to 14 pounds
391                 8 to 9 inches tall at the shoulder    4 to 6 pounds

          Life Span  Adaptability  Adapts Well To Apartment Living  \
0    10 to 12 years           NaN                                1
1    13 to 18 years           NaN                                4
2    12 to 14 years           NaN                                5
3    10 to 12 years           NaN                                5
4    10 to 13 years           NaN                                1
..              …            …                              …
387  10 to 14 years           NaN                                1
388  14 to 20 years           NaN                                5
389  10 to 12 years           NaN                                1
390  10 to 15 years           NaN                                5
391  12 to 15 years           NaN                                5

     Good For Novice Owners  Sensitivity Level  Tolerates Being Alone  …  \
0                         1                  3                      3  …
1                         4                  4                      1  …
2                         4                  3                      1  …
3                         3                  5                      2  …
4                         2                  3                      2  …
..                      …                 …                    …  …
387                       3                  4                      1  …
```

```
388                    1                  5                      1  …
389                    3                  3                      1  …
390                    5                  4                      3  …
391                    4                  5                      2  …

     Potential For Mouthiness  Prey Drive  Tendency To Bark Or Howl  \
0                           4         4.0                       4.0
1                           4         2.0                       4.0
2                           4         3.0                       2.0
3                           3         5.0                       2.0
4                           5         5.0                       4.0
..                        …         …                       …
387                         3         4.0                       4.0
388                         3         5.0                       5.0
389                         2         3.0                       4.0
390                         3         3.0                       5.0
391                         2         2.0                       3.0

     Wanderlust Potential  Physical Needs  Energy Level  Intensity  \
0                       4             NaN             4          4
1                       2             NaN             4          3
2                       2             NaN             4          3
3                       5             NaN             5          2
4                       4             NaN             5          3
..                    …             …           …        …
387                     4             NaN             5          3
388                     5             NaN             3          3
389                     3             NaN             4          3
390                     2             NaN             5          3
391                     3             NaN             5          4

     Exercise Needs  Potential For Playfulness  \
0                 4                          3
1                 3                          3
2                 3                          4
3                 4                          4
4                 5                          5
..              …                        …
387               4                          5
388               3                          3
389               5                          4
390               3                          4
391               4                          5

                          Detailed Description Link
0              https://dogtime.com/dog-breeds/afador
1         https://dogtime.com/dog-breeds/affenhuahua
```

```
2         https://dogtime.com/dog-breeds/affenpinscher
3          https://dogtime.com/dog-breeds/afghan-hound
4      https://dogtime.com/dog-breeds/airedale-terrier
..                                                  …
387  https://dogtime.com/dog-breeds/wirehaired-poin…
388       https://dogtime.com/dog-breeds/xoloitzuintli
389      https://dogtime.com/dog-breeds/yakutian-laika
390            https://dogtime.com/dog-breeds/yorkipoo
391   https://dogtime.com/dog-breeds/yorkshire-terrier

[392 rows x 37 columns]
```

[4]:
```
df.isna().sum() #Check for NaN's
```

[4]:
```
Unnamed: 0                            0
Dog Breed Group                       0
Height                                3
Weight                                7
Life Span                             0
Adaptability                        392
Adapts Well To Apartment Living       0
Good For Novice Owners                0
Sensitivity Level                     0
Tolerates Being Alone                 0
Tolerates Cold Weather                0
Tolerates Hot Weather                 0
All Around Friendliness             392
Affectionate With Family              0
Kid-Friendly                          0
Dog Friendly                          0
Friendly Toward Strangers             0
Health And Grooming Needs           392
Amount Of Shedding                    0
Drooling Potential                    1
Easy To Groom                         0
General Health                        0
Potential For Weight Gain             0
Size                                  0
Trainability                        392
Easy To Train                         0
Intelligence                          0
Potential For Mouthiness              0
Prey Drive                            1
Tendency To Bark Or Howl              1
Wanderlust Potential                  0
Physical Needs                      392
Energy Level                          0
```

```
Intensity                            0
Exercise Needs                       0
Potential For Playfulness            0
Detailed Description Link            0
dtype: int64
```

```
[5]: #dimensionality reduction to filter out columns with all NA values
     df.drop(['Unnamed: 0', 'Adaptability', 'All Around Friendliness', 'Health And␣
      ↪Grooming Needs',
            'Trainability', 'Physical Needs','Detailed Description Link'], axis=1,␣
      ↪inplace=True)
```

```
[6]: #drop remaining tuples with na values
     df = df.dropna(subset=['Height', 'Weight','Drooling Potential', 'Tendency To␣
      ↪Bark Or Howl', 'Prey Drive'])
```

```
[7]: #regex to convert height, weight, and life span to numbers, also to turn dog␣
      ↪breed group to categorical numbers
     df = df.copy()

     df['Height'] = df['Height'].apply(lambda x: (int(re.search(r'(\d+)\s*foot', x).
      ↪group(1)) if re.search(r'(\d+)\s*foot', x) else 0) * 12 +
                                              int(re.search(r'(\d+)\s*inch', x).
      ↪group(1)) if re.search(r'(\d+)\s*inch', x) else 0)
     df['Weight'] = df['Weight'].apply(lambda x: (sum(map(int, re.findall(r'\d+',␣
      ↪x))) / 2) if re.search(r'\d+\s*to\s*\d+\s*pounds', x) else 0)

     df['Life Span'] = df['Life Span'].apply(lambda x: (sum(map(int, re.
      ↪findall(r'\d+', x))) / 2) if re.search(r'\d+\s*to\s*\d+\s*years', x) else 0)

     df['Dog Breed Group'] = LabelEncoder().fit_transform(df['Dog Breed Group'])
```

```
[8]: olddf = df.copy() #create copy of dataframe before using aggregation
```

```
[9]: #Aggregate columns using average

     df['Adaptability'] = df.iloc[:,4:10].mean(axis=1) #adaptability
     df['Friendliness'] = df.iloc[:,10:14].mean(axis=1) #friendliness
     df['HealthGrooming'] = df.iloc[:,14:20].mean(axis=1) #health and grooming needs
     df['Trainability'] = df.iloc[:,20:26].mean(axis=1) #Training
     df['Physical'] = df.iloc[:,26:].mean(axis=1) #physical need
```

```
[10]: #get rid of unexamined columns in new dataframe

      df.drop(['Adapts Well To Apartment Living', 'Good For Novice Owners',
             'Sensitivity Level', 'Tolerates Being Alone', 'Tolerates Cold Weather',
             'Tolerates Hot Weather', 'Affectionate With Family', 'Kid-Friendly',
             'Dog Friendly', 'Friendly Toward Strangers', 'Amount Of Shedding',
             'Drooling Potential', 'Easy To Groom', 'General Health',
             'Potential For Weight Gain', 'Size', 'Easy To Train',
             'Potential For Mouthiness', 'Prey Drive', 'Tendency To Bark Or Howl',
             'Wanderlust Potential', 'Energy Level', 'Intensity', 'Exercise Needs',
             'Potential For Playfulness'], axis=1, inplace=True)
```

[11]:
```
olddf #before agg
```

[11]:
```
     Dog Breed Group  Height  Weight  Life Span  \
0                  4      29    62.5       11.0
1                  4      12     8.0       15.5
2                  0      11     8.0       13.0
3                  2      26    55.0       11.0
4                  6      23    52.5       11.5
..               …       …      …         …
387                5      24    55.0       12.0
388                0      18    30.0       17.0
389                7      23    47.5       11.0
390                3      15     8.5       12.5
391                0       9     5.0       13.5

     Adapts Well To Apartment Living  Good For Novice Owners  \
0                                  1                       1
1                                  4                       4
2                                  5                       4
3                                  5                       3
4                                  1                       2
..                               …                      …
387                                1                       3
388                                5                       1
389                                1                       3
390                                5                       5
391                                5                       4

     Sensitivity Level  Tolerates Being Alone  Tolerates Cold Weather  \
0                    3                      3                       4
1                    4                      1                       2
2                    3                      1                       3
3                    5                      2                       5
4                    3                      2                       3
..                 …                    …                     …
387                  4                      1                       4
388                  5                      1                       3
389                  3                      1                       5
390                  4                      3                       2
391                  5                      2                       2
```

```
     Tolerates Hot Weather  …  Easy To Train  Intelligence  \
0                        2  …              1             5   
1                        3  …              3             3   
2                        3  …              2             4   
3                        5  …              1             4   
4                        3  …              4             5   
..                     …  …            …           …   
387                      3  …              5             5   
388                      3  …              3             5   
389                      3  …              4             4   
390                      3  …              4             4   
391                      2  …              3             3   

     Potential For Mouthiness  Prey Drive  Tendency To Bark Or Howl  \
0                           4         4.0                       4.0   
1                           4         2.0                       4.0   
2                           4         3.0                       2.0   
3                           3         5.0                       2.0   
4                           5         5.0                       4.0   
..                        …         …                       …   
387                         3         4.0                       4.0   
388                         3         5.0                       5.0   
389                         2         3.0                       4.0   
390                         3         3.0                       5.0   
391                         2         2.0                       3.0   

     Wanderlust Potential  Energy Level  Intensity  Exercise Needs  \
0                       4             4          4               4   
1                       2             4          3               3   
2                       2             4          3               3   
3                       5             5          2               4   
4                       4             5          3               5   
..                    …           …        …             …   
387                     4             5          3               4   
388                     5             3          3               3   
389                     3             4          3               5   
390                     2             5          3               3   
391                     3             5          4               4   

     Potential For Playfulness  
0                            3  
1                            3  
2                            4  
3                            4  
4                            5  
..                         …  
```

```
387                          5
388                          3
389                          4
390                          4
391                          5

[382 rows x 30 columns]
```

[12]:
```
olddf.columns
```

[12]:
```
Index(['Dog Breed Group', 'Height', 'Weight', 'Life Span',
       'Adapts Well To Apartment Living', 'Good For Novice Owners',
       'Sensitivity Level', 'Tolerates Being Alone', 'Tolerates Cold Weather',
       'Tolerates Hot Weather', 'Affectionate With Family', 'Kid-Friendly',
       'Dog Friendly', 'Friendly Toward Strangers', 'Amount Of Shedding',
       'Drooling Potential', 'Easy To Groom', 'General Health',
       'Potential For Weight Gain', 'Size', 'Easy To Train', 'Intelligence',
       'Potential For Mouthiness', 'Prey Drive', 'Tendency To Bark Or Howl',
       'Wanderlust Potential', 'Energy Level', 'Intensity', 'Exercise Needs',
       'Potential For Playfulness'],
      dtype='object')
```

[13]:
```
df #after agg
```

[13]:
```
     Dog Breed Group  Height  Weight  Life Span  Intelligence  Adaptability  \
0                  4      29    62.5       11.0             5      2.333333
1                  4      12     8.0       15.5             3      3.000000
2                  0      11     8.0       13.0             4      3.166667
3                  2      26    55.0       11.0             4      4.166667
4                  6      23    52.5       11.5             5      2.333333
..               …       …       …          …             …             …
387                5      24    55.0       12.0             5      2.666667
388                0      18    30.0       17.0             5      3.000000
389                7      23    47.5       11.0             4      2.666667
390                3      15     8.5       12.5             4      3.666667
391                0       9     5.0       13.5             3      3.333333

     Friendliness  HealthGrooming  Trainability  Physical
0            2.50        3.333333      3.666667  3.354167
1            2.75        2.833333      3.000000  3.072917
2            3.25        2.166667      2.833333  3.177083
3            4.00        2.333333      3.333333  3.604167
4            3.75        2.500000      4.500000  3.885417
..              …             …             …        …
387          4.75        2.500000      4.166667  3.885417
388          2.75        3.000000      4.333333  3.135417
389          4.25        3.166667      3.333333  3.677083
```

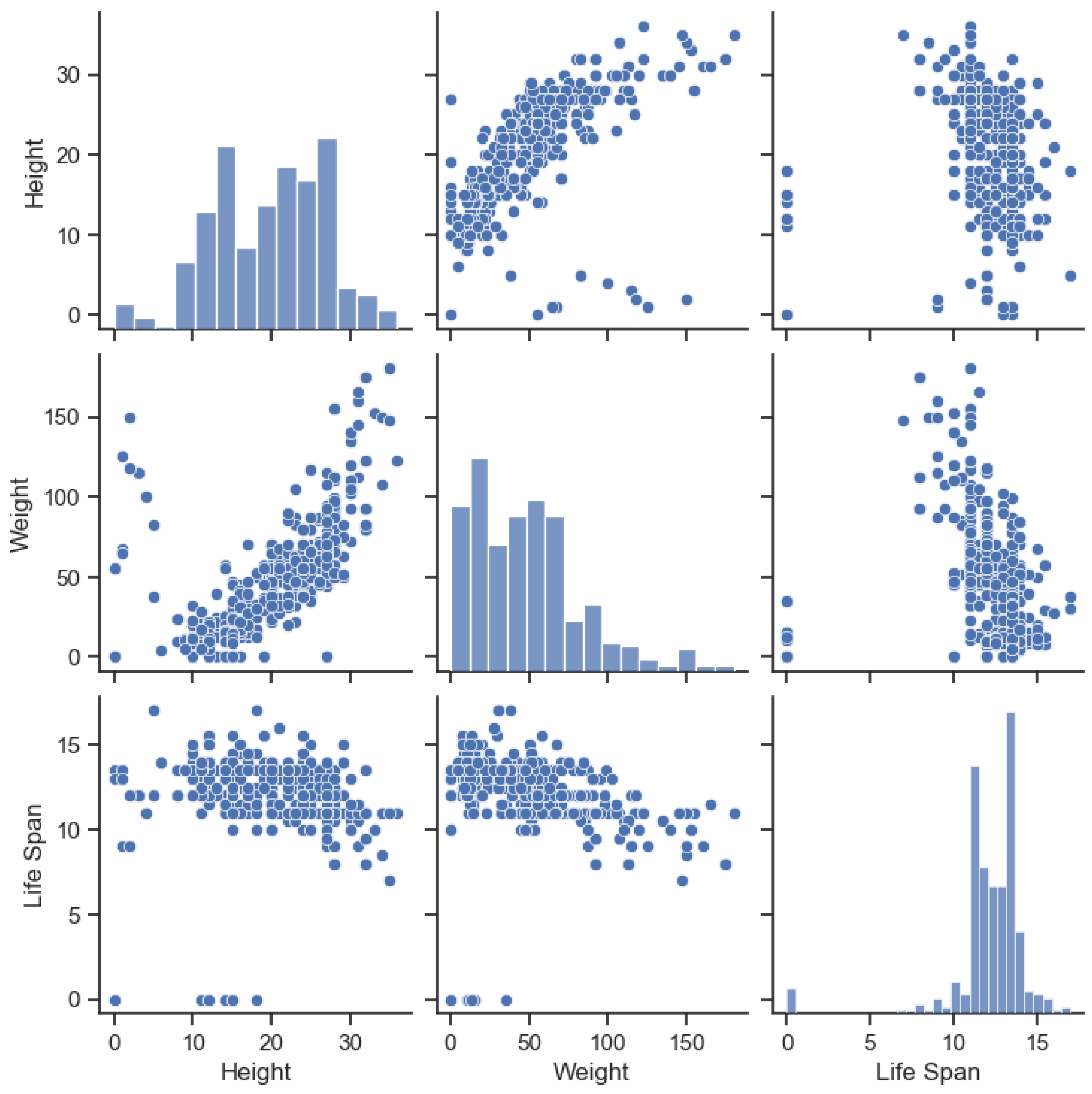
```
390          3.50        2.000000      3.500000  3.458333
391          2.75        2.000000      2.666667  3.593750

[382 rows x 10 columns]
```

[14]:
```
df.columns
```

[14]:
```
Index(['Dog Breed Group', 'Height', 'Weight', 'Life Span', 'Intelligence',
       'Adaptability', 'Friendliness', 'HealthGrooming', 'Trainability',
       'Physical'],
      dtype='object')
```

1.1.1 After some preprocessing, the preview of data will give insights about distributions/correlations. We can see the attributes weighed against one-another as scatterplots in a SPLOM (Scatterplot Matrix). All diagonal entries show distributions.

[15]:
```
sns.set(style="ticks")
sns.pairplot(df[['Height', 'Weight', 'Life Span']], height=2.5)
plt.show()
```



```
[16]: np.corrcoef(df['Height'],df['Weight'])[0,1]
```

```
[16]: 0.6629259674738956
```

```
[17]: np.corrcoef(df['Weight'],df['Life Span'])[0,1]
```

```
[17]: -0.24899745634939424
```

```
[18]: np.corrcoef(df['Height'],df['Life Span'])[0,1]
```

```
[18]: -0.1209275924298133
```

1.1.2 Because height, weight, and life span seem to have some correlation, we can cluster them into size classes with K means clustering

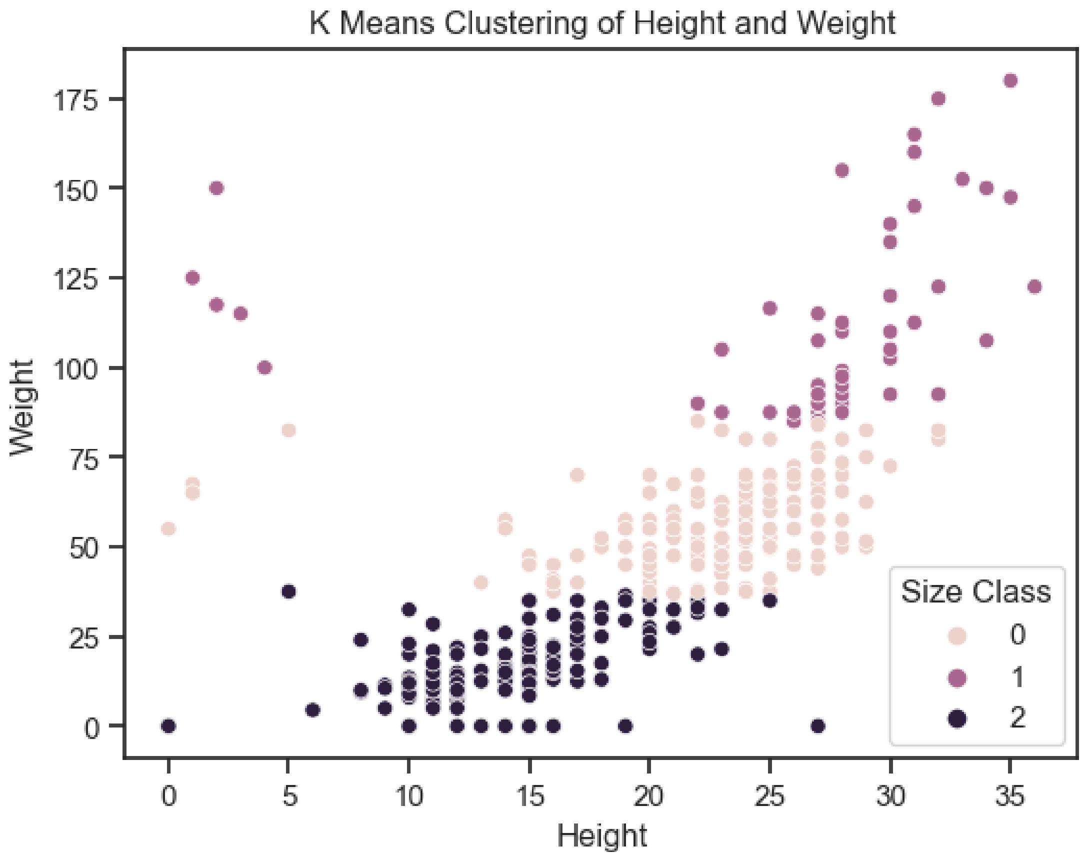
[19]:
```
#kmeans results make new column
kmeans = KMeans(n_clusters=3)
olddf['Size Class'] = kmeans.fit_predict(df[['Height', 'Weight']])
df['Size Class Exp'] = kmeans.fit_predict(df[['Height', 'Weight']])
```

[20]:
```
#kmeans results make new column
kmeans = KMeans(n_clusters=3)
df['Size Class'] = kmeans.fit_predict(df[['Height', 'Weight', 'Life Span']])
```

[21]:
```
df['Size Class'].unique()
```

[21]:
```
array([0, 1, 2])
```

[22]:
```
sns.scatterplot(x='Height', y='Weight', hue='Size Class', data=olddf,)
plt.title('K Means Clustering of Height and Weight')
plt.show()
```



```
[23]: fig = px.scatter_3d(df[['Height', 'Weight', 'Life Span', 'Size Class']],␣
      ↪x='Height', y='Weight', z='Life Span',
                    color='Size Class', title='K Means Clustering of Height, Weight,␣
      ↪and Life Span')

      fig.show()
```

1.2 Model Creation and Analysis

1.2.1 Now that we have a cleaned dataframe, we can try to make the best fit model. Three sets of attributes will create different predictions of intelligence based on different attributes examined. Each set will posses the following:

1.2.2 Set 1. Examines all attributes originally, before any aggregation and clustering

1.2.3 Set 2. Examines the averages of the original attributes, without clustering

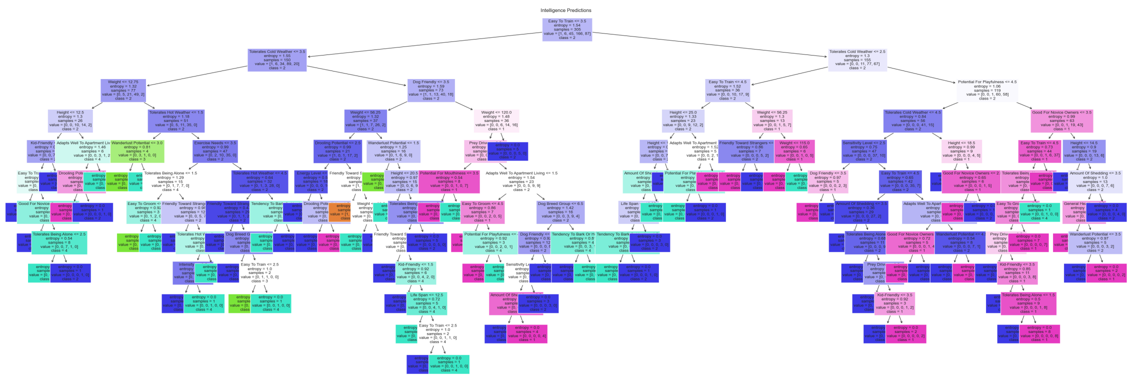
1.2.4 Set 3. Examines the averages of the orignal attributes, with height and weight clustered (K=3)

1.2.5 Set 4. Examines the averages of the original attributes, with height, weight, and life span clustered (K=3)

1.3 Set 1

```
[24]: #Make Trained Decision Tree for 'DF' to predict intelligence class

      #grab attributes and class attribute
      allAtr = olddf[['Dog Breed Group', 'Height', 'Weight', 'Life Span',
             'Adapts Well To Apartment Living', 'Good For Novice Owners',
             'Sensitivity Level', 'Tolerates Being Alone', 'Tolerates Cold Weather',
             'Tolerates Hot Weather', 'Affectionate With Family', 'Kid-Friendly',
             'Dog Friendly', 'Friendly Toward Strangers', 'Amount Of Shedding',
             'Drooling Potential', 'Easy To Groom', 'General Health',
             'Potential For Weight Gain', 'Easy To Train',
             'Potential For Mouthiness', 'Prey Drive', 'Tendency To Bark Or Howl',
             'Wanderlust Potential', 'Energy Level', 'Intensity', 'Exercise Needs',
             'Potential For Playfulness', 'Size Class']]
      classAtr = olddf['Intelligence']

      #partition 20% of data to be tested, map to xtr-Training Attributes, xt-Test␣
      ↪Attributes, ytr-Class Training Attributes, yt-Class Test Attributes (used␣
      ↪for accuracy)
      xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr,␣
      ↪test_size=0.2, random_state=42)

      #instantiates tree object
      AllTree = tree.DecisionTreeClassifier(criterion="entropy")

      #make the tree
```

```
AllTree.fit(xtr, ytr)

#predict based on test data
prediction = AllTree.predict(xt)

#plot data
plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=[str(i) for i␣
 ↪in df['Intelligence'].unique()], filled=True, impurity=True, precision=2,␣
 ↪fontsize=10)
plt.title('Intelligence Predictions')
plt.show()
```

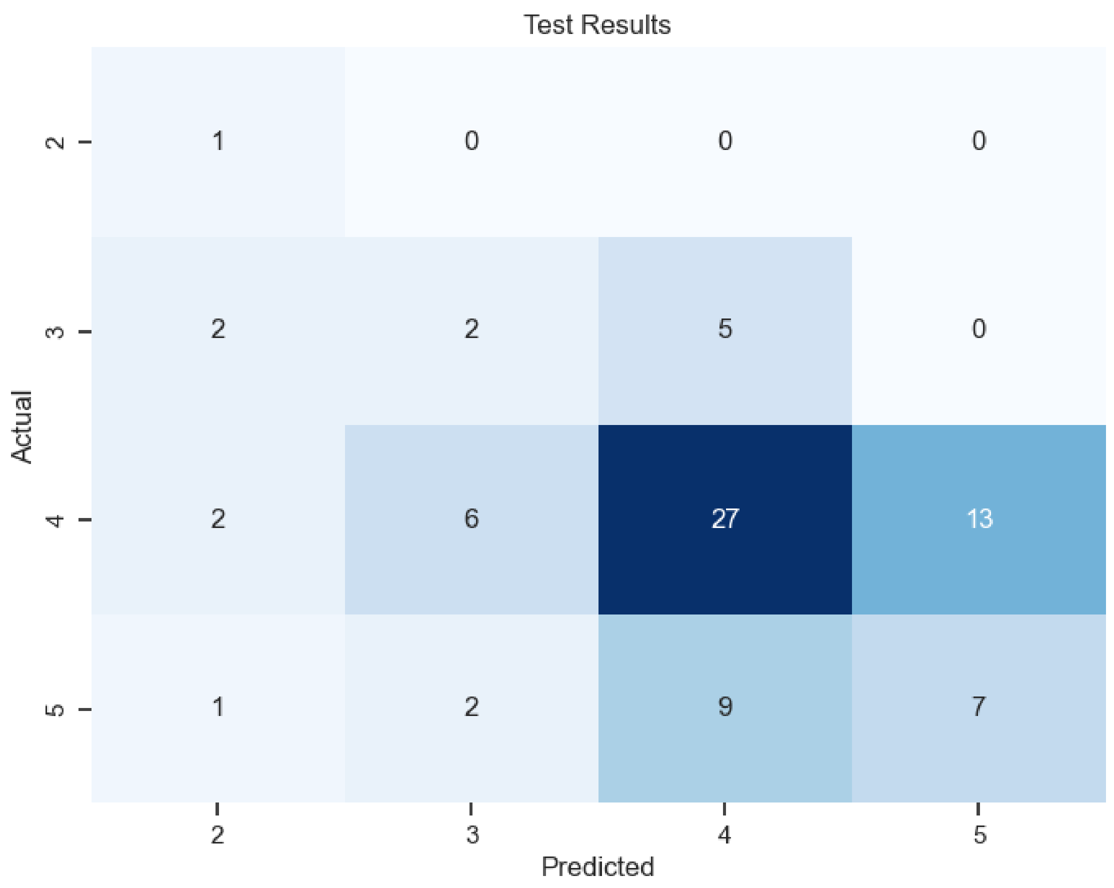


```
[25]: importances = AllTree.feature_importances_ #get gain of each attribute␣
       ↪examined, put in list

      print("Gain:")

      #Print the attributes in the order of importance
      for i in (-importances).argsort():
          print(f'Attribute: {xtr.columns[i]}, Importance: {importances[i]:.2f}')
```

```
Gain:
Attribute: Easy To Train, Importance: 0.14
Attribute: Tolerates Cold Weather, Importance: 0.08
Attribute: Weight, Importance: 0.08
Attribute: Height, Importance: 0.07
Attribute: Good For Novice Owners, Importance: 0.05
Attribute: Tolerates Being Alone, Importance: 0.05
Attribute: Potential For Playfulness, Importance: 0.05
Attribute: Adapts Well To Apartment Living, Importance: 0.05
Attribute: Dog Friendly, Importance: 0.05
Attribute: Friendly Toward Strangers, Importance: 0.04
Attribute: Tolerates Hot Weather, Importance: 0.04
```

```
Attribute: Wanderlust Potential, Importance: 0.04
Attribute: Kid-Friendly, Importance: 0.03
Attribute: Amount Of Shedding, Importance: 0.03
Attribute: Drooling Potential, Importance: 0.03
Attribute: Dog Breed Group, Importance: 0.03
Attribute: Prey Drive, Importance: 0.03
Attribute: Easy To Groom, Importance: 0.02
Attribute: Tendency To Bark Or Howl, Importance: 0.02
Attribute: Sensitivity Level, Importance: 0.02
Attribute: Exercise Needs, Importance: 0.01
Attribute: Life Span, Importance: 0.01
Attribute: Potential For Mouthiness, Importance: 0.01
Attribute: Energy Level, Importance: 0.01
Attribute: General Health, Importance: 0.01
Attribute: Intensity, Importance: 0.01
Attribute: Potential For Weight Gain, Importance: 0.00
Attribute: Affectionate With Family, Importance: 0.00
Attribute: Size Class, Importance: 0.00
```

[26]:
```
cm = metrics.confusion_matrix(yt, prediction)

unique_classes = sorted(set(yt) & set(prediction))

#Create df
conf_matrix_df = pd.DataFrame(cm, index=unique_classes, columns=unique_classes)

#Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Test Results')
plt.show()
```



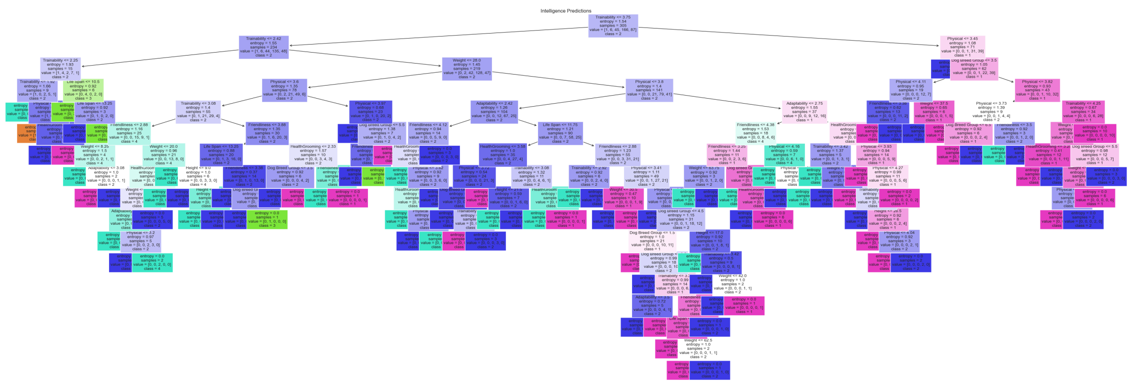
[27]:
```
#Calculate performance metrics
accuracy = metrics.accuracy_score(yt, prediction)
precision = metrics.precision_score(yt, prediction, average='macro')
recall = metrics.recall_score(yt, prediction, average='macro')
f1 = metrics.f1_score(yt, prediction, average='macro')

#Print performance metrics
print("Accuracy for Intelligence Prediction:", accuracy)
print("Equation for accuracy: (TP + TN) / All")
print("\nPrecision, what % of tuples predicted positive were correct:",␣
 ↪precision)
print("Equation for precision: TP / (TP + FP)")
print("\nRecall, what % of tuples were classified as positive:", recall)
print("Equation for recall: TP / (TP + FN)")
print("\nF measure, measures balance between both precision and recall␣
 ↪(harmonic mean):", f1)
print("Equation for F measure: (2 * precision * recall) / (precision + recall)")
```

```
Accuracy for Intelligence Prediction: 0.4805194805194805
```

```
Equation for accuracy: (TP + TN) / All

Precision, what % of tuples predicted positive were correct: 0.34380081300813015
Equation for precision: TP / (TP + FP)

Recall, what % of tuples were classified as positive: 0.5382858187134503
Equation for recall: TP / (TP + FN)

F measure, measures balance between both precision and recall (harmonic mean):
0.3654891333779565
Equation for F measure: (2 * precision * recall) / (precision + recall)
```

1.4 Set 2

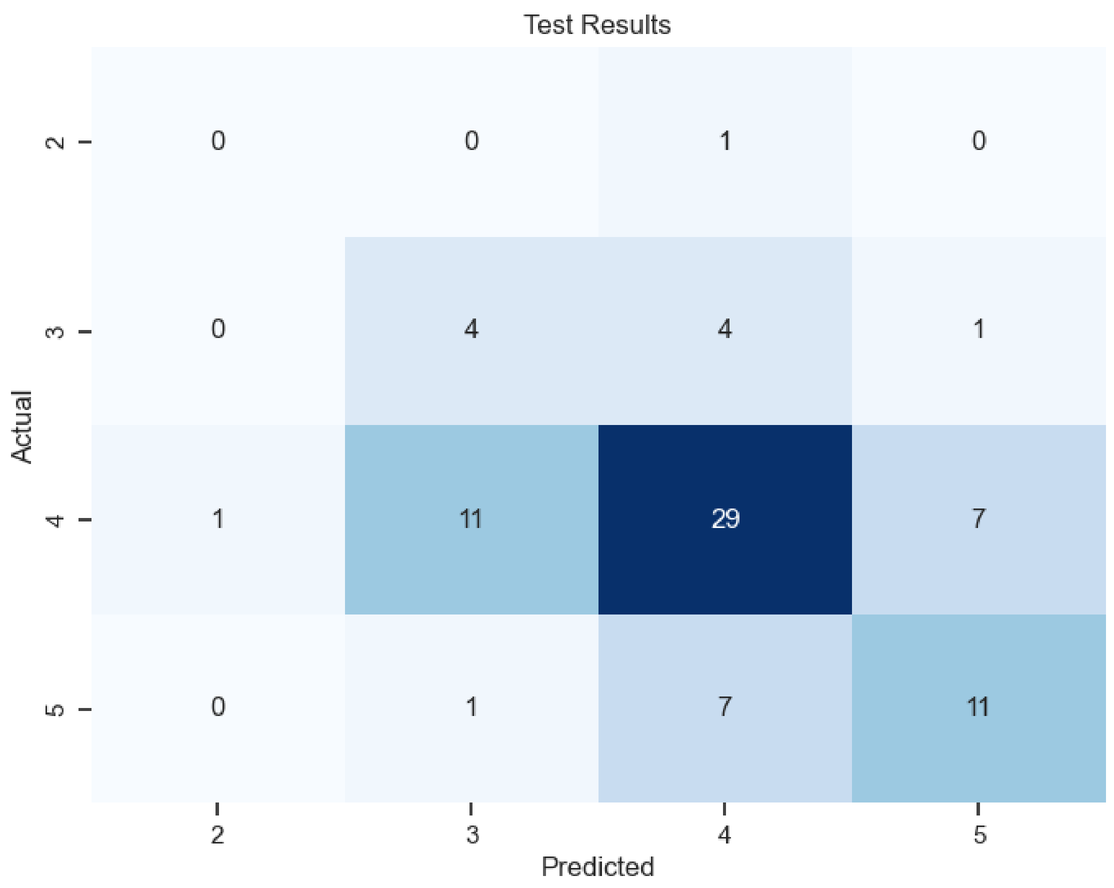
[28]:
```python
allAtr = df[['Dog Breed Group', 'Height', 'Weight', 'Life Span',
        'Adaptability', 'Friendliness', 'HealthGrooming', 'Trainability',
        'Physical']]
classAtr = df['Intelligence']

xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr,␣
 ↪test_size=0.2, random_state=42)

AllTree = tree.DecisionTreeClassifier(criterion="entropy")

AllTree.fit(xtr, ytr)

prediction = AllTree.predict(xt)

plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=[str(i) for i␣
 ↪in df['Intelligence'].unique()], filled=True, impurity=True, precision=2,␣
 ↪fontsize=10)
plt.title('Intelligence Predictions')
plt.show()
```



[29]:
```
importances = AllTree.feature_importances_

print("Gain:")

for i in (-importances).argsort():
    print(f'Attribute: {xtr.columns[i]}, Importance: {importances[i]:.2f}')
```

```
Gain:
Attribute: Trainability, Importance: 0.22
Attribute: Physical, Importance: 0.21
Attribute: Friendliness, Importance: 0.12
Attribute: Dog Breed Group, Importance: 0.11
Attribute: Weight, Importance: 0.11
Attribute: HealthGrooming, Importance: 0.09
Attribute: Adaptability, Importance: 0.05
Attribute: Life Span, Importance: 0.04
Attribute: Height, Importance: 0.04
```

[30]:
```
cm = metrics.confusion_matrix(yt, prediction)

unique_classes = [2,3,4,5]

conf_matrix_df = pd.DataFrame(cm, index=unique_classes, columns=unique_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Test Results')
plt.show()
```



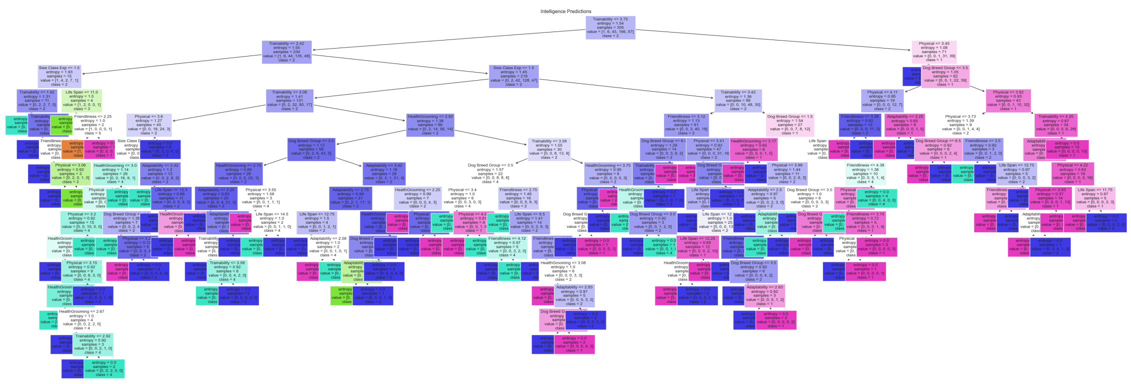
```
[31]: accuracy = metrics.accuracy_score(yt, prediction)
precision = metrics.precision_score(yt, prediction, average='macro')
recall = metrics.recall_score(yt, prediction, average='macro')
f1 = metrics.f1_score(yt, prediction, average='macro')

print("Accuracy for Intelligence Prediction:", accuracy)
print("Equation for accuracy: (TP + TN) / All")
print("\nPrecision, what % of tuples predicted positive were correct:",␣
 ↪precision)
print("Equation for precision: TP / (TP + FP)")
print("\nRecall, what % of tuples were classified as positive:", recall)
print("Equation for recall: TP / (TP + FN)")
print("\nF measure, measures balance between both precision and recall␣
 ↪(harmonic mean):", f1)
print("Equation for F measure: (2 * precision * recall) / (precision + recall)")
```

```
Accuracy for Intelligence Prediction: 0.5714285714285714
Equation for accuracy: (TP + TN) / All
```

```
Precision, what % of tuples predicted positive were correct: 0.3840661103979461
Equation for precision: TP / (TP + FP)

Recall, what % of tuples were classified as positive: 0.40688961988304095
Equation for recall: TP / (TP + FN)

F measure, measures balance between both precision and recall (harmonic mean):
0.3876581904198699
Equation for F measure: (2 * precision * recall) / (precision + recall)
```

1.5 Set 3

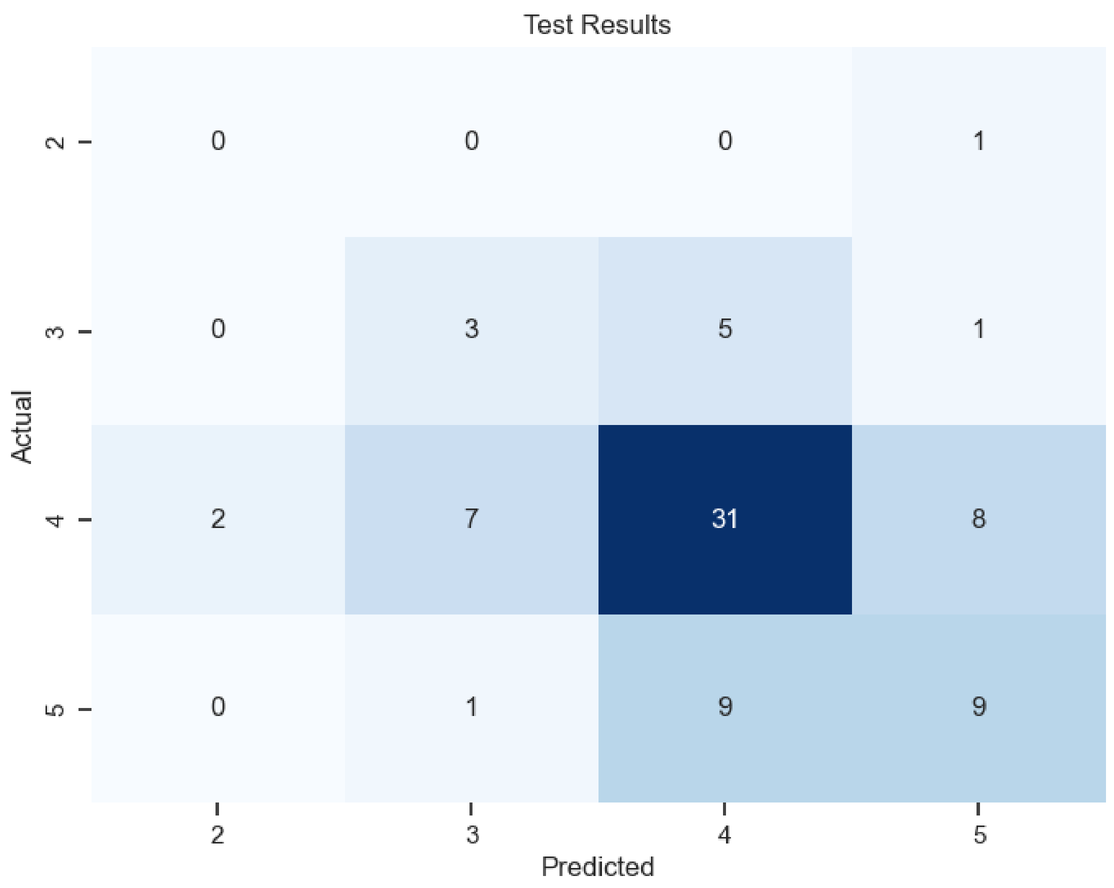
[32]:
```python
allAtr = df[['Dog Breed Group', 'Size Class Exp', 'Life Span',
        'Adaptability', 'Friendliness', 'HealthGrooming', 'Trainability',
        'Physical']]
classAtr = df['Intelligence']

xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr,␣
 ↪test_size=0.2, random_state=42)

AllTree = tree.DecisionTreeClassifier(criterion="entropy")

AllTree.fit(xtr, ytr)

prediction = AllTree.predict(xt)

plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=[str(i) for i␣
 ↪in df['Intelligence'].unique()], filled=True, impurity=True, precision=2,␣
 ↪fontsize=10)
plt.title('Intelligence Predictions')
plt.show()
```



[33]:
```
importances = AllTree.feature_importances_

print("Gain:")

for i in (-importances).argsort():
    print(f'Attribute: {xtr.columns[i]}, Importance: {importances[i]:.2f}')
```

```
Gain:
Attribute: Trainability, Importance: 0.21
Attribute: Physical, Importance: 0.18
Attribute: Dog Breed Group, Importance: 0.14
Attribute: Life Span, Importance: 0.11
Attribute: Adaptability, Importance: 0.11
Attribute: HealthGrooming, Importance: 0.10
Attribute: Friendliness, Importance: 0.10
Attribute: Size Class Exp, Importance: 0.06
```

[34]:
```
cm = metrics.confusion_matrix(yt, prediction)

unique_classes = sorted(set(yt) & set(prediction))

conf_matrix_df = pd.DataFrame(cm, index=unique_classes, columns=unique_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Test Results')
plt.show()
```



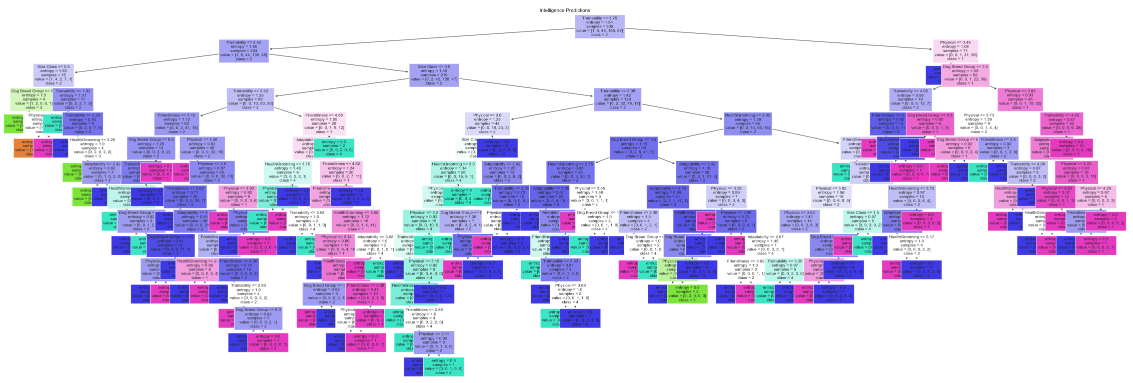
[35]:
```
accuracy = metrics.accuracy_score(yt, prediction)
precision = metrics.precision_score(yt, prediction, average='macro')
recall = metrics.recall_score(yt, prediction, average='macro')
f1 = metrics.f1_score(yt, prediction, average='macro')

print("Accuracy for Intelligence Prediction:", accuracy)
print("Equation for accuracy: (TP + TN) / All")
print("\nPrecision, what % of tuples predicted positive were correct:",␣
 ↪precision)
print("Equation for precision: TP / (TP + FP)")
print("\nRecall, what % of tuples were classified as positive:", recall)
print("Equation for recall: TP / (TP + FN)")
print("\nF measure, measures balance between both precision and recall␣
 ↪(harmonic mean):", f1)
print("Equation for F measure: (2 * precision * recall) / (precision + recall)")
```

```
Accuracy for Intelligence Prediction: 0.5584415584415584
Equation for accuracy: (TP + TN) / All
```

```
Precision, what % of tuples predicted positive were correct: 0.3588250930356193
Equation for precision: TP / (TP + FP)

Recall, what % of tuples were classified as positive: 0.3632127192982456
Equation for recall: TP / (TP + FN)

F measure, measures balance between both precision and recall (harmonic mean):
0.3600877192982456
Equation for F measure: (2 * precision * recall) / (precision + recall)
```

1.6 Set 4

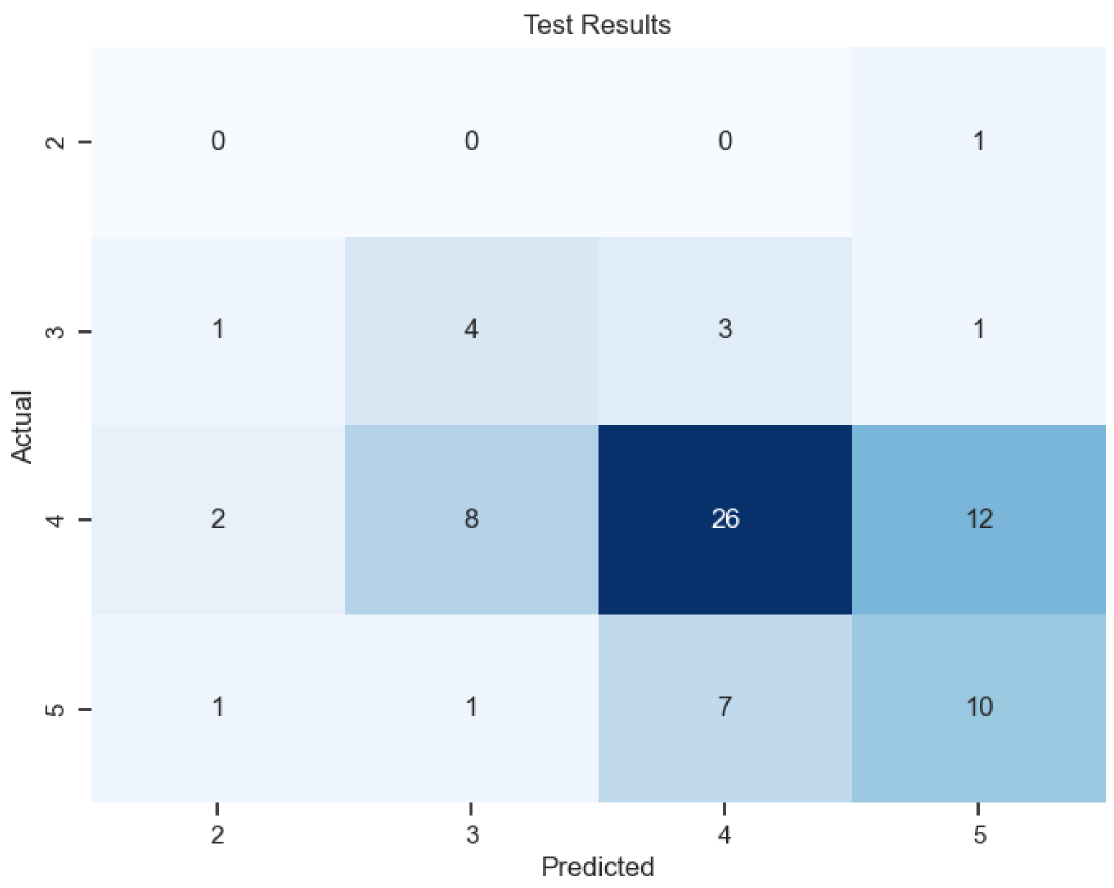
```
[36]: allAtr = df[['Dog Breed Group','Adaptability', 'Friendliness',
                   'HealthGrooming', 'Trainability', 'Physical', 'Size Class']]
      classAtr = df['Intelligence']

      xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr,␣
       ↪test_size=0.2, random_state=42)

      AllTree = tree.DecisionTreeClassifier(criterion="entropy")

      AllTree.fit(xtr, ytr)

      prediction = AllTree.predict(xt)

      plt.figure(figsize=(48, 16))
      tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=[str(i) for i␣
       ↪in df['Intelligence'].unique()], filled=True, impurity=True, precision=2,␣
       ↪fontsize=10)
      plt.title('Intelligence Predictions')
      plt.show()
```



[37]:
```
importances = AllTree.feature_importances_

print("Gain:")

for i in (-importances).argsort():
    print(f'Attribute: {xtr.columns[i]}, Importance: {importances[i]:.2f}')
```

```
Gain:
Attribute: Trainability, Importance: 0.23
Attribute: Physical, Importance: 0.23
Attribute: Dog Breed Group, Importance: 0.13
Attribute: HealthGrooming, Importance: 0.13
Attribute: Friendliness, Importance: 0.11
Attribute: Adaptability, Importance: 0.10
Attribute: Size Class, Importance: 0.07
```

[38]:
```
cm = metrics.confusion_matrix(yt, prediction)

unique_classes = sorted(set(yt) & set(prediction))

conf_matrix_df = pd.DataFrame(cm, index=unique_classes, columns=unique_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Test Results')
plt.show()
```



[39]:
```
accuracy = metrics.accuracy_score(yt, prediction)
precision = metrics.precision_score(yt, prediction, average='macro')
recall = metrics.recall_score(yt, prediction, average='macro')
f1 = metrics.f1_score(yt, prediction, average='macro')

print("Accuracy for Intelligence Prediction:", accuracy)
print("Equation for accuracy: (TP + TN) / All")
print("\nPrecision, what % of tuples predicted positive were correct:",␣
 ↪precision)
print("Equation for precision: TP / (TP + FP)")
print("\nRecall, what % of tuples were classified as positive:", recall)
print("Equation for recall: TP / (TP + FN)")
print("\nF measure, measures balance between both precision and recall␣
 ↪(harmonic mean):", f1)
print("Equation for F measure: (2 * precision * recall) / (precision + recall)")
```

```
Accuracy for Intelligence Prediction: 0.5194805194805194
Equation for accuracy: (TP + TN) / All
```

```
Precision, what % of tuples predicted positive were correct: 0.36164529914529914
Equation for precision: TP / (TP + FP)

Recall, what % of tuples were classified as positive: 0.3781067251461988
Equation for recall: TP / (TP + FN)

F measure, measures balance between both precision and recall (harmonic mean):
0.3619500654384375
Equation for F measure: (2 * precision * recall) / (precision + recall)
```

1.7 Conclusion

1.7.1 After utilizing various preprocessing techniques, we can see that set 2, which uses aggregation and no clustering, has the highest prediction accuracy; however, this cannot be generalized to other datasets. Every data set is independent and should be explored thoroughly before determining what preprocessing is required for analysis. It is important to experiment with various data preprocessing techniques to weigh their influence in the final prediction model.