a2-python-5831

● Graded

Group

Tagore Kosireddy
Mihret Kemal
Michael Ngala
...and 1 more
View or edit group

Total Points

229.75 / 236 pts

Autograder Score

142.5 / 148.0

Failed Tests

Public Tests
q0 (1/2)
q1e (3.5/4)
q4 (13/16)
q6d (3/4)

Passed Tests

q1b (3/3)
q1c (4/4)
q1d (3/3)
q1f (5/5)
q1g (3/3)
q1h (4/4)
q2 (30/30)
q3 (10/10)
q6a (3/3)
q6b (4/4)
q6c (6/6)
q6e (5/5)
q6f (6/6)
q6g (10/10)
q6h (10/10)
q6i (10/10)
q6j (6/6)

Question 2

Question 1 — **31.25** / 32 pts

2.1 **Q1a** — **2** / 2 pts

- ✔ **– 0 pts** Correct

2.2 **Q1i** — **6** / 6 pts

- ✔ **– 0 pts** Correct

2.3 **Q1j** — **5.5** / 6 pts

- ✔ **– 0.5 pts** x_ticks (violins) not ordered by mpaa_rating

2.4 **Q1k** — **5.5** / 6 pts

- ✔ **– 0.5 pts** missing legend / overlapping

2.5 **Q1l** — **5.25** / 6 pts

- ✔ **– 0.75 pts** x-axis range should be limited or clipped to 0 to 100.
- ✔ **– 0 pts** legend items reversed

2.6 **Q1m** — **3** / 6 pts

- ✔ **– 3 pts** wrong plot

2.7 **Q1bonus** — **4** / 0 pts

- ✔ **+ 4 pts** Correct

Question 3

Question 2 — **10** / 10 pts

- ✔ **– 0 pts** Correct

Question 4

Question 3 — **6** / 6 pts

- ✔ **– 0 pts** Correct

Question 5

Question 4 — **6** / 6 pts

- ✔ **– 0 pts** Correct

Question 6

Question 5

6 / 6 pts

✔ **– 0 pts** Correct

Question 7

Question 6

18 / 18 pts

7.1 **Q6h**

6 / 6 pts

✔ **– 0 pts** Correct

7.2 **Q6i**

12 / 12 pts

✔ **– 0 pts** Correct

Question 8

GENERAL

10 / 10 pts

✔ **– 0 pts** Correct

Autograder Results

Autograder Output

```
 _______      __        __
/ _____ \   __|  |__   __|  |__
|  /    \ |  |__   __|  |__   __|  ______   _ _____
| |      | |    |  |       |  |    |  ___  |  | |/ ___|
| |      | |    |  |       |  |    | |___| |  |   /
| |      | |    |  |       |  |    | _____|  |  |
|  \_____/ |    |  |_      |  |_   | |_____  |  |
 \________/     \__|      \__|    \______|  |__|
                                    v5.2.3
```

---------- GRADING SUMMARY ----------

Score for q0 (1.000) differs from logged score (2.000)
Score for q2 (21.000) differs from logged score (30.000)
Score for q3 (4.000) differs from logged score (10.000)
Score for q6f (3.000) differs from logged score (6.000)

Total Score: 142.500 / 148.000 (96.284%)

	name	score	max_score
0	Public Tests	NaN	NaN
1	q0	1.0	2.0
2	q1b	3.0	3.0
3	q1c	4.0	4.0
4	q1d	3.0	3.0
5	q1e	3.5	4.0
6	q1f	5.0	5.0
7	q1g	3.0	3.0
8	q1h	4.0	4.0
9	q2	30.0	30.0
10	q3	10.0	10.0
11	q4	13.0	16.0
12	q6a	3.0	3.0
13	q6b	4.0	4.0
14	q6c	6.0	6.0
15	q6d	3.0	4.0
16	q6e	5.0	5.0
17	q6f	6.0	6.0
18	q6g	10.0	10.0
19	q6h	10.0	10.0

20	q6i	10.0	10.0
21	q6j	6.0	6.0

Public Tests

```
q0 results:
    q0 - 1 result:
        Test case passed

    q0 - 2 result:
        Test case failed
        Trying:
            LLM == True | GS == True
        Expecting:
            True
        **********************************************************************
        Line 1, in q0 1
        Failed example:
            LLM == True | GS == True
        Expected:
            True
        Got:
            False

q1b results: All test cases passed!

q1c results: All test cases passed!

q1d results: All test cases passed!

q1e results: All test cases passed!

q1f results: All test cases passed!

q1g results: All test cases passed!

q1h results: All test cases passed!

q2 results: All test cases passed!

q3 results: All test cases passed!

q4 results: All test cases passed!

q6a results: All test cases passed!

q6b results: All test cases passed!

q6c results: All test cases passed!

q6d results: All test cases passed!

q6e results: All test cases passed!
```

q6f results: All test cases passed!

q6g results: All test cases passed!

q6h results: All test cases passed!

q6i results: All test cases passed!

q6j results: All test cases passed!

q0 (1/2)

```
q0 results:
    q0 - 1 result:
        Test case passed

    q0 - 2 result:
        Test case failed
        Trying:
            LLM == True | GS == True
        Expecting:
            True
        **********************************************************************
        Line 1, in q0 1
        Failed example:
            LLM == True | GS == True
        Expected:
            True
        Got:
            False
```

q1b (3/3)

q1b results: All test cases passed!

q1c (4/4)

q1c results: All test cases passed!

q1d (3/3)

q1d results: All test cases passed!

q1e (3.5/4)

```
q1e results:
    q1e - 1 result:
        Test case passed

    q1e - 2 result:
        Test case passed

    q1e - 3 result:
        Test case passed

    q1e - 4 result:
        Test case passed

    q1e - 5 result:
        Test case passed

    q1e - 6 result:
        Test case passed

    q1e - 7 result:
        Test case passed

    q1e - 8 result:
        Test case passed

    q1e - 9 result:
        Test case passed

    q1e - 10 result:
        Test case passed

    q1e - 11 result:
        Test case passed

    q1e - 12 result:
        Test case passed

    q1e - 13 result:
        Test case passed

    q1e - 14 result:
        Test case failed
      Trying:
          type_imdb_rating == 4
      Expecting:
          True
      **********************************************************************
      Line 1, in q1e 13
```

```
    Failed example:
        type_imdb_rating == 4
    Expected:
        True
    Got:
        False

q1e - 15 result:
    Test case passed

q1e - 16 result:
    Test case passed
```

q1f (5/5)

q1f results: All test cases passed!

q1g (3/3)

q1g results: All test cases passed!

q1h (4/4)

q1h results: All test cases passed!

q2 (30/30)

q2 results: All test cases passed!

q3 (10/10)

q3 results: All test cases passed!

q4 (13/16)

```
q4 results:
    q4 - 1 result:
        Test case passed

    q4 - 2 result:
        Test case passed

    q4 - 3 result:
        Test case passed

    q4 - 4 result:
        Test case passed

    q4 - 5 result:
        Test case passed

    q4 - 6 result:
        Test case failed
      Trying:
          all(round(perfDF['MCC'], 6) == [0.333333, 0.5, 0.218218, 0.408248, 0.6, 0.408248, 0.654654, 0.5, 0.33333
      Expecting:
          True
      **********************************************************************
      Line 1, in q4 5
      Failed example:
          all(round(perfDF['MCC'], 6) == [0.333333, 0.5, 0.218218, 0.408248, 0.6, 0.408248, 0.654654, 0.5, 0.33333
      Expected:
          True
      Got:
          False
```

q6a (3/3)

q6a results: All test cases passed!

q6b (4/4)

q6b results: All test cases passed!

q6c (6/6)

q6c results: All test cases passed!

q6d (3/4)

```
q6d results:
    q6d - 1 result:
        Test case passed

    q6d - 2 result:
        Test case failed
        Trying:
            q6d_ii == 1
        Expecting:
            True
        **********************************************************************
        Line 1, in q6d 1
        Failed example:
            q6d_ii == 1
        Expected:
            True
        Got:
            False

    q6d - 3 result:
        Test case passed
```

q6e (5/5)

q6e results: All test cases passed!

q6f (6/6)

q6f results: All test cases passed!

q6g (10/10)

q6g results: All test cases passed!

q6h (10/10)

q6h results: All test cases passed!

q6i (10/10)

q6i results: All test cases passed!

q6j (6/6)
q6j results: All test cases passed!

Submitted Files

a2 - python - CS5831

This assignment will cover topics from data, data preprocessing, and classification.

Make sure that you keep this notebook named as "a2-5831.ipynb"

Submit the zip-file created after running your notebook on the Linux lab machines.

Any other packages or tools, outside those listed in the assignments or Canvas, should be cleared by Dr. Brown before use in your submission.

Q0 - Setup

The following code looks to see whether your notebook is run on Gradescope (GS), Colab (COLAB), or the linux Python environment you were asked to setup.

In [1]:
```
import re
import os
import platform
import sys

# flag if notebook is running on Gradescope
if re.search(r'amzn', platform.uname().release):
    GS = True
else:
    GS = False

# flag if notebook is running on Colaboratory
try:
  import google.colab
  COLAB = True
except:
  COLAB = False

# flag if running on Linux lab machines.
cname = platform.uname().node
if re.search(r'(guardian|colossus|c28|coc-15954-m)', cname):
    LLM = True
else:
```

```
    LLM = False

print("System: GS - %s, COLAB - %s, LLM - %s" % (GS, COLAB, LLM))
```

```
System: GS - False, COLAB - False, LLM - True
```

Notebook Setup

It is good practice to list all imports needed at the top of the notebook. You can import modules in later cells as needed, but listing them at the top clearly shows all which are needed to be available / installed.

If you are doing development on Colab, the otter-grader package is not available, so you will need to install it with pip (uncomment the cell directly below).

In [2]:
```
# Only uncomment if you developing on Colab
# if COLAB == True:
#     print("Installing otter:")
#     !pip install otter-grader==4.2.0
```

In [3]:
```
# Import standard DS packages
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import math
import statistics
import textwrap
%matplotlib inline

from sklearn import tree        # decision tree classifier
from sklearn import neighbors   # knn classifier
from sklearn import naive_bayes # naive bayes classifier
from sklearn import metrics     # performance evaluation metrics
from sklearn import model_selection
from sklearn import preprocessing
from sklearn import pipeline
# import graphviz, pydotplus

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline, make_pipeline
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score,mean_squared_error
from sklearn.metrics import confusion_matrix

# Package for Autograder
import otter
grader = otter.Notebook()

from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

In [4]:
```
grader.check("q0")
```

Out [4]: q0 results: All test cases passed!

Q1 - Exploratory Data Analysis

Consider the `movies` data set available on Canvas. The data set is made up of over 600 randomly selected movies, released before 2016, with information extracted from IMDB and Rotten Tomatoes. A code book on the variables is also provided.

You should explore the files a bit in a text editor to understand the format. The variables are made up of different types: nominal, ordinal, numeric, etc. We will refer to the different variables by their column / codebook names.

Missing Data

Q1(a) - Examine data for loading

Look at the `movies` data set. Is there any missing data in the `movies` data set? If so, how is it encoded?

Missing data is encoded as: There are some missing values in the movies dataset that are encoded as NA.

Q1(b) - Load the data

Load the movies data into a DataFrame `q1movies`. Is there any missing data in the `movies` data set? If yes, make sure to encode those missing values when reading the data in pandas `read_csv` function.

In [37]:
```
# Read in movies data with pandas "read_csv" function
#  Use column names from the original csv file

q1movies = pd.read_csv('movies.csv', na_values = ['NA'])

q1movies.head()
```

Out [37]:

	title	title_type	genre	runtime	mpaa_rating
0	Filly Brown	Feature Film	Drama	80.0	R
1	The Dish	Feature Film	Drama	101.0	PG-13
2	Waiting for Guffman	Feature Film	Comedy	84.0	R
3	The Age of Innocence	Feature Film	Drama	139.0	PG
4	Malevolence	Feature Film	Horror	90.0	R

	studio	thtr_rel_year	thtr_rel_month	thtr_rel_day
0	Indomina Media Inc.	2013	4	19
1	Warner Bros. Pictures	2001	3	14
2	Sony Pictures Classics	1996	8	21
3	Columbia Pictures	1993	10	1
4	Anchor Bay Entertainment	2004	9	10

	dvd_rel_year	...	best_dir_win	top200_box	director
0	2013.0	...	no	no	Michael D. Olmos
1	2001.0	...	no	no	Rob Sitch
2	2001.0	...	no	no	Christopher Guest
3	2001.0	...	yes	no	Martin Scorsese
4	2005.0	...	no	no	Stevan Mena

	actor1	actor2	actor3
0	Gina Rodriguez	Jenni Rivera	Lou Diamond Phillips
1	Sam Neill	Kevin Harrington	Patrick Warburton
2	Christopher Guest	Catherine O'Hara	Parker Posey
3	Daniel Day-Lewis	Michelle Pfeiffer	Winona Ryder
4	Samantha Dark	R. Brandon Johnson	Brandon Johnson

	actor4	actor5
0	Emilio Rivera	Joseph Julian Soria
1	Tom Long	Genevieve Mooy
2	Eugene Levy	Bob Balaban
3	Richard E. Grant	Alec McCowen
4	Heather Magee	Richard Glover

```
                                imdb_url  \
0  http://www.imdb.com/title/tt1869425/
1  http://www.imdb.com/title/tt0205873/
2  http://www.imdb.com/title/tt0118111/
3  http://www.imdb.com/title/tt0106226/
4  http://www.imdb.com/title/tt0388230/

                                      rt_url
0      //www.rottentomatoes.com/m/filly_brown_2012/
1                //www.rottentomatoes.com/m/dish/
2   //www.rottentomatoes.com/m/waiting_for_guffman/
3      //www.rottentomatoes.com/m/age_of_innocence/
4  //www.rottentomatoes.com/m/10004684-malevolence/

[5 rows x 32 columns]
```

In [38]:
```
grader.check("q1b")
```

Out [38]:
```
q1b results: All test cases passed!
```

Q1(c) - Missing data

We want to understand where (which variable) and how much data is missing (for each variable the percentage of rows).

In [39]:
```
# Create a Data.Series that has the percentage of missing data for each
#  Calculate percentage of missing values and report
miss_data = (q1movies.isnull().sum()/len(q1movies))*100
miss_data
```

Out [39]:
```
title             0.000000
title_type        0.000000
genre             0.000000
runtime           0.154799
mpaa_rating       0.000000
studio            1.083591
thtr_rel_year     0.000000
thtr_rel_month    0.000000
thtr_rel_day      0.000000
dvd_rel_year      1.238390
dvd_rel_month     1.238390
dvd_rel_day       1.238390
imdb_rating       0.000000
```

```
imdb_num_votes      0.000000
critics_rating      0.000000
critics_score       0.000000
audience_rating     0.000000
audience_score      0.000000
best_pic_nom        0.000000
best_pic_win        0.000000
best_actor_win      0.000000
best_actress_win    0.000000
best_dir_win        0.000000
top200_box          0.000000
director            0.309598
actor1              0.309598
actor2              1.083591
actor3              1.393189
actor4              1.857585
actor5              2.167183
imdb_url            0.000000
rt_url              0.000000
dtype: float64
```

In [40]:
```
grader.check("q1c")
```

Out [40]:
```
q1c results: All test cases passed!
```

Q1(d) - Clean data

We want to clean up data with respect to the missing values.

Ignore any missing values in the `studio`, `dvd_rel_year`, `dvd_rel_month`, `dvd_rel_day`, and all variables including and listed after `best_pic_nom`. For other missing values, remove the sample that contains the missing value.

Save the resulting DataFrame in the `movies` variable.

In [41]:
```
# Ignore missing values in "studio", "dvd_rel_year", "dvd_rel_month",
#   "dvd_rel_day", and all varaibles including and after "best_pic_nom"
# For other missing values, remove the sample that contains the missing value.

# select the columns to ignore missing values
col_ignore =[ 'studio','dvd_rel_year','dvd_rel_month','dvd_rel_day']
# finding the index of best_pic_nom
best_pic_nom_index = q1movies.columns.get_loc('best_pic_nom')
# merge the columns listed after best_pic_nom with the columns to ignore
```

```
col_ignore.extend(q1movies.columns[best_pic_nom_index:])
# select columns which are not in columns to ignore
considered_cols= [col for col in q1movies.columns if col not in col_ignore]
# drop missing values from the considered columns
movies = q1movies.dropna(subset = considered_cols)

movies.shape
```

Out [41]: (645, 32)

In [42]:
```
grader.check("q1d")
```

Out [42]: q1d results: All test cases passed!

Q1(e) - Attribute Types

For the following variables, state the attribute type: 1- *nominal*, 2- *ordinal*, 3- *interval*, or 4- *ratio*.

In [43]:
```
# Attribute types of variables
type_genre = 1
type_runtime = 4
type_mpaa_rating = 2
type_studio = 1
type_thtr_rel_month = 2
type_imdb_rating = 2
type_audience_score = 4
type_best_pic_win = 1
```

In [44]:
```
grader.check("q1e")
```

Out [44]: q1e results: All test cases passed!

Summary Statistics

Q1(f) - Statistics, part 1

For the following variables, report out a five number summary: `audience_score` and `imdb_rating`

Store results in a DataFrame: `q1f`

Hint: consider using the `describe` function.

In [47]:
```
# Report five number summary for variables `audience_score` and `imdb_rating` in
#  a DataFrame "q1f"
#  Rows should represent: min, Q1 - 25%, Q2 - 50%, Q3 - 50%, max
#  Columns should be `audience_score` then `imdb_rating`
# finding the summary for specific columns
summary = q1movies[['audience_score','imdb_rating']].describe()
# considering only five number summary that we want and store on q1f
q1f = summary.loc[['min','25%','50%','75%','max']]
```

In [48]:
```
grader.check("q1f")
```

Out [48]: q1f results: All test cases passed!

Q1(g) - Statistics, part 2

Report the mean, median, and mode of `critics_score` and `runtime` to the given variables.

In [49]:
```
# Report mean, median and mode of "critics_score" and "runtime"

# For critics_score-
q1g_cs_mean = round(movies['critics_score'].mean(),4)
q1g_cs_median = round(movies['critics_score'].median(),4)
q1g_cs_mode = round(movies['critics_score'].mode()[0],4)

# For runtime-
q1g_r_mean = round(movies['runtime'].mean(),4)
q1g_r_median = round(movies['runtime'].median(),4)
q1g_r_mode = round(movies['runtime'].mode()[1],4)
```

In [50]:
```
grader.check("q1g")
```

Out [50]: q1g results: All test cases passed!

Q1(h) - Statistics, part 3

Report the first quartile, 37th percentile, third quartile, and 83rd percentile for `critics_score` and `runtime` and assign it to the given variables.

In [51]:
```
# Report first quartile, 31st percentile, third quartile, and 90th percentile
#  of "critics_score" and "runtime"

# For critics_score-
q1h_cs_q1 = np.quantile(movies['critics_score'], .25)
q1h_cs_p37 = np.percentile(movies['critics_score'], 37)
q1h_cs_q3 = np.quantile(movies['critics_score'], .75)
q1h_cs_p83 = np.percentile(movies['critics_score'], 83)

# For runtime-
q1h_r_q1 = np.quantile(movies['runtime'], .25)
q1h_r_p37 = np.percentile(movies['runtime'], 37)
q1h_r_q3 = np.quantile(movies['runtime'], .75)
q1h_r_p83 = np.percentile(movies['runtime'], 83)
```

In [52]:
```
grader.check("q1h")
```

Out [52]:
```
q1h results: All test cases passed!
```

Visualizations

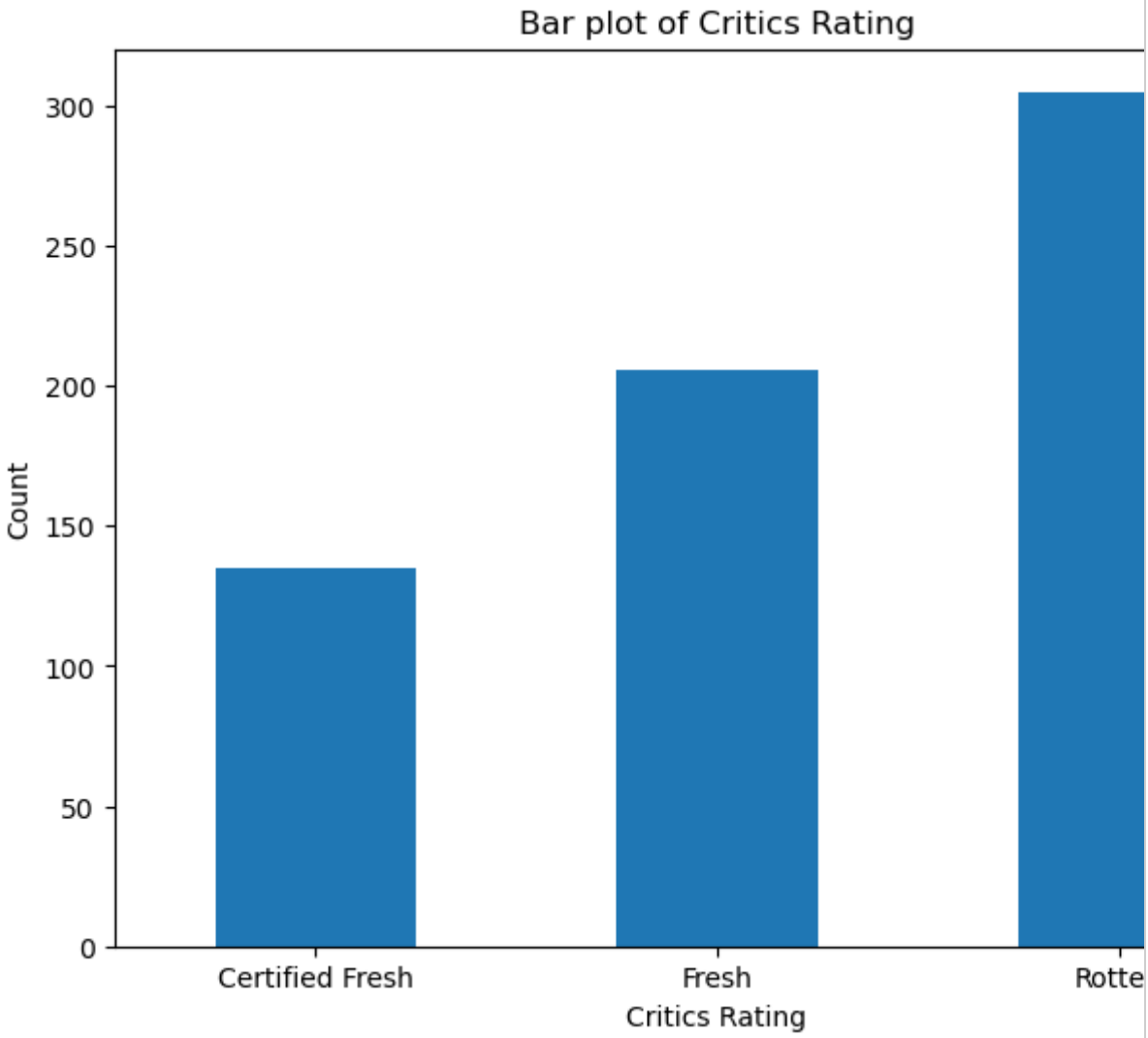
Q1(i) Visualization: Single Variable

I highly encourage looking at the Fundamentals of Visualization reference book to guide in the creation of “good” visualizations requested below.

Create a bar plot for the `critics_rating` variable.

In [19]:
```
# Create bar plot for "critics_rating"
plt.figure(figsize=(8,6))
freq_table = q1movies.groupby('critics_rating').apply(len)
freq_table.plot(kind = 'bar')

# Set labels and title
plt.title('Bar plot of Critics Rating')
plt.xticks(rotation = 0)
plt.xlabel('Critics Rating')
plt.ylabel('Count')
```

```
plt.show()
```



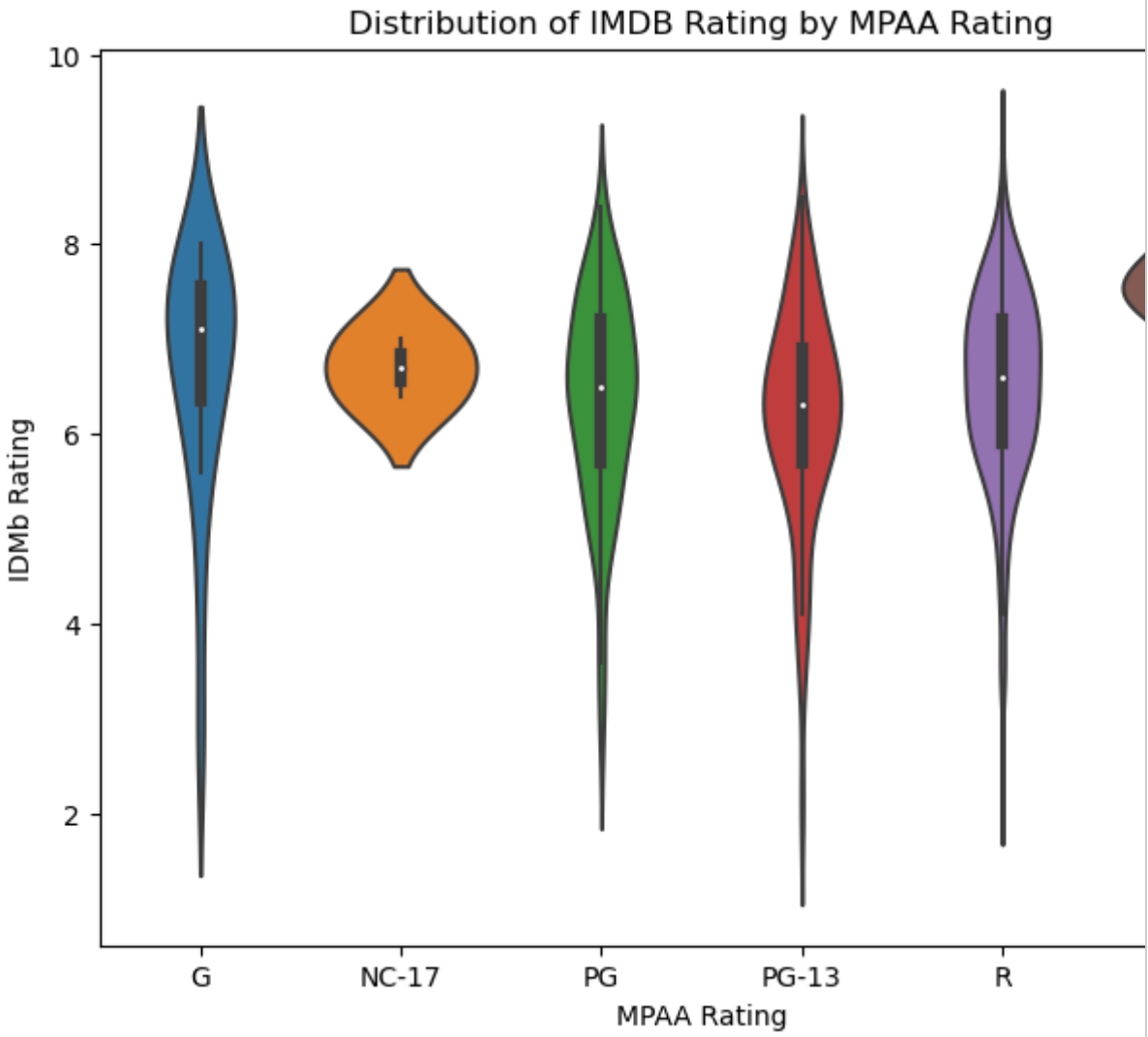
Q1(j) Visualization: Two Variables

Create a violin plot for `imdb_rating` grouped by `mpaa_rating` (sorted by mpaa rating, where 'Unrated' is last).

In [20]:

```
# Create a violin plot for `imdb_rating` grouped by `mpaa_rating` (sorted)
sorted_mpaa_rating = sorted(movies['mpaa_rating'].unique(),key = lambda x: (x == 'Unrated',x))
plt.figure(figsize = (8,6))
sns.violinplot(x = 'mpaa_rating',y ='imdb_rating', data = movies, order = sorted_mpaa_rating)

# Set labels and title
plt.title('Distribution of IMDB Rating by MPAA Rating')
```

```
plt.xlabel('MPAA Rating')
plt.ylabel('IDMb Rating')
plt.show()
```



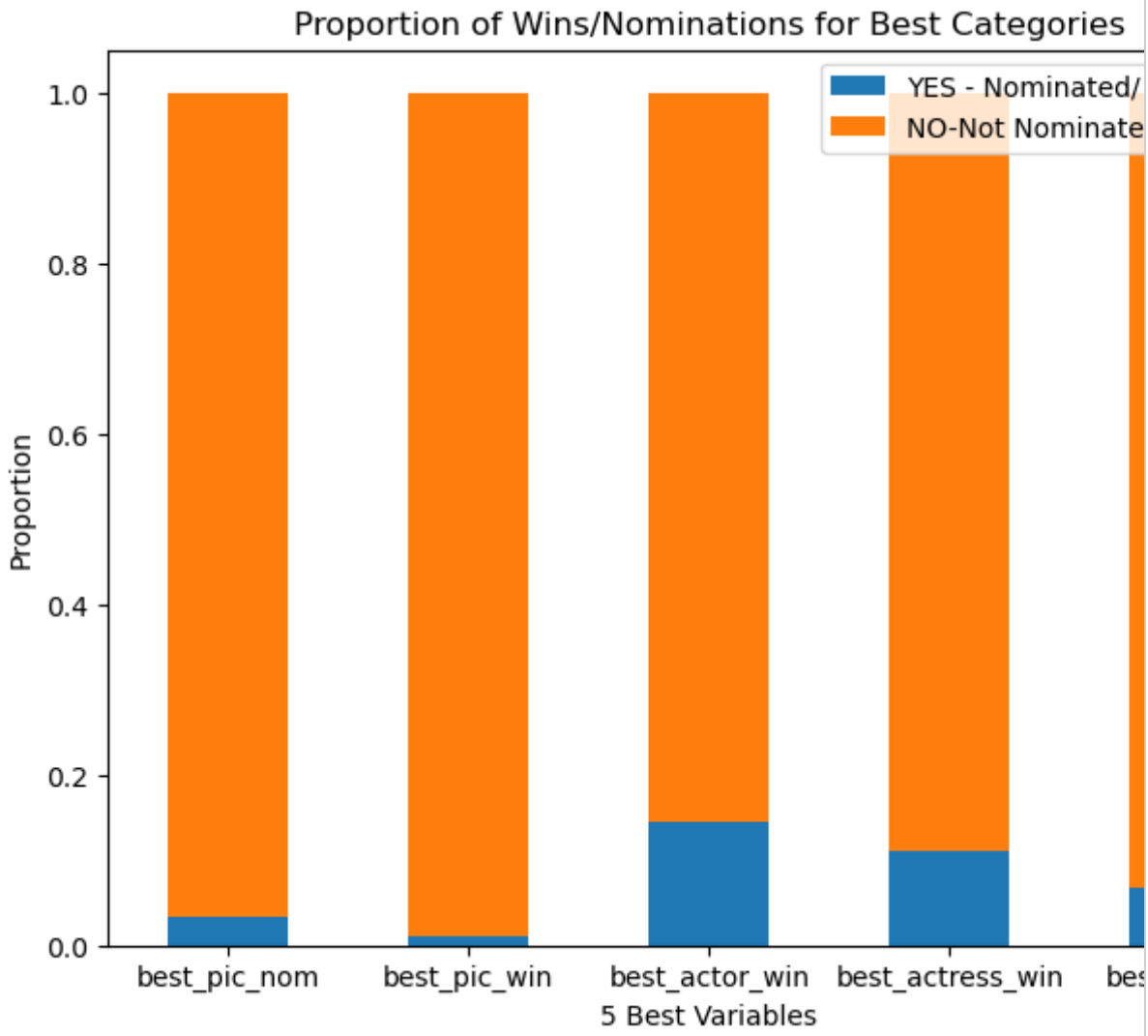
Q1(k) Visualization: Multiple variables

Create a stacked bar chart to display the proportion of wins (nominations) for the 5 `best_*` variables.

In [21]:

```
# Create a stacked bar chart to display the proportion of wins (nominations)
#   for the 5 `best_*` variables.

# storing the 5 best_*columns
best_vars = ['best_pic_nom','best_pic_win', 'best_actor_win',
             'best_actress_win','best_dir_win']
# maping the values in the column with yes to 1 and rest to 0
df_best = movies[best_vars].applymap (lambda x: 1 if x == 'yes' else 0)
```

```
# find sum for each category to get counts of yes
best_counts = df_best.sum()
# Calculate proportions of yes
best_proportions = best_counts / len(movies)
# Creating a DataFrame for the proportions of both yes and no
df_plot = pd.DataFrame({'YES - Nominated/ Win': best_proportions,
                        'NO-Not Nominated/ Not Win': 1 - best_proportions})
# Plotting the stacked bar chart
df_plot.plot(kind ='bar', stacked = True, figsize = (8, 6))

# Set labels and title
plt.title('Proportion of Wins/Nominations for Best Categories')
plt.xlabel('5 Best Variables')
plt.ylabel('Proportion' )
plt.xticks(rotation = 0)
plt.legend()
plt.show()
```

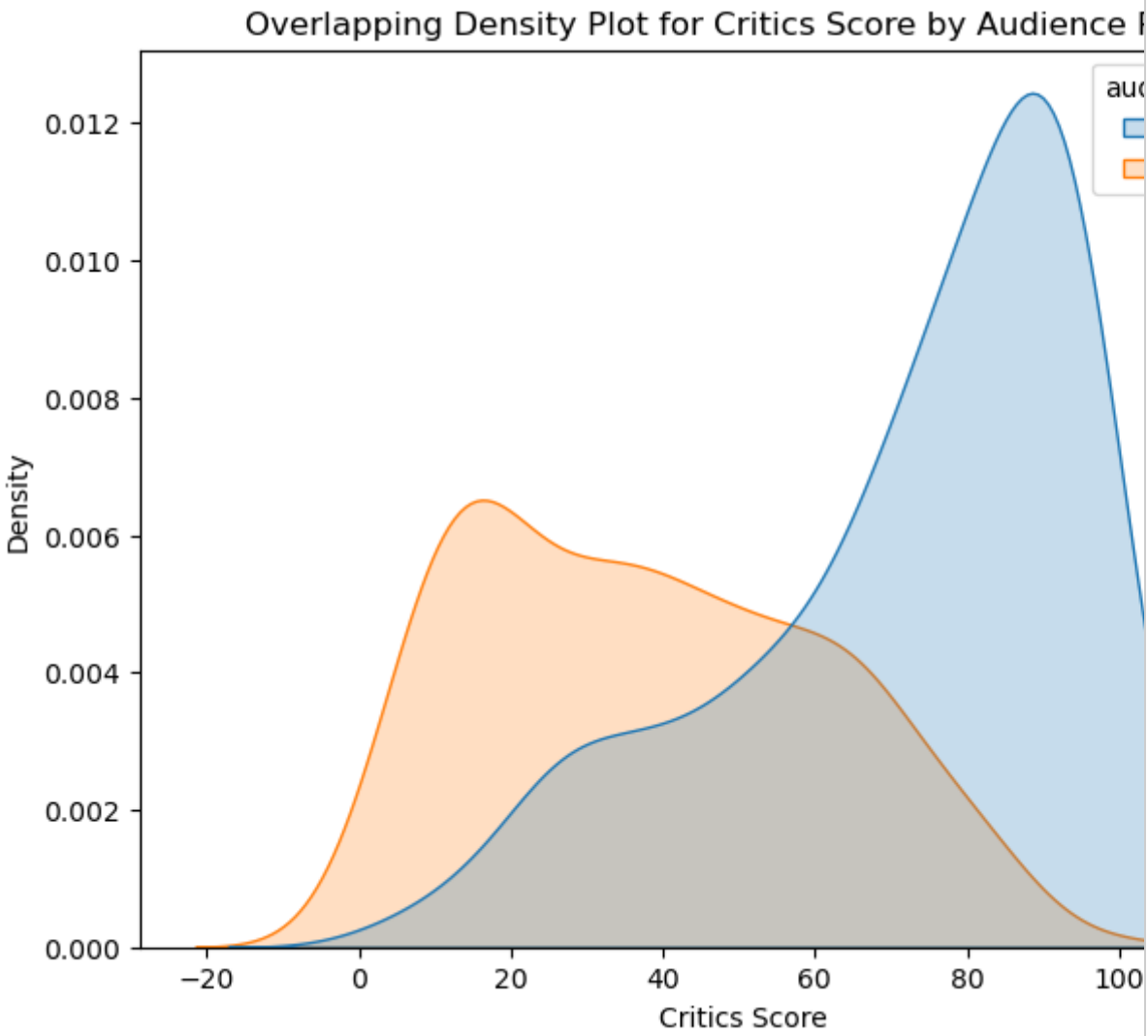


Q1(l) Visualization: Two Variables

Create an overlapping density plot for `critics_score` grouped by `audience_rating`.

In [22]:

```
# Create overlapping density plot
plt.figure(figsize = (8, 6))
sns.kdeplot(data = movies, x = 'critics_score', hue = 'audience_rating', fill =
True,palette = 'tab10')

# Set labels and title
plt.xlabel('Critics Score')
plt.ylabel('Density')
plt.title('Overlapping Density Plot for Critics Score by Audience Rating')
plt.show()
```

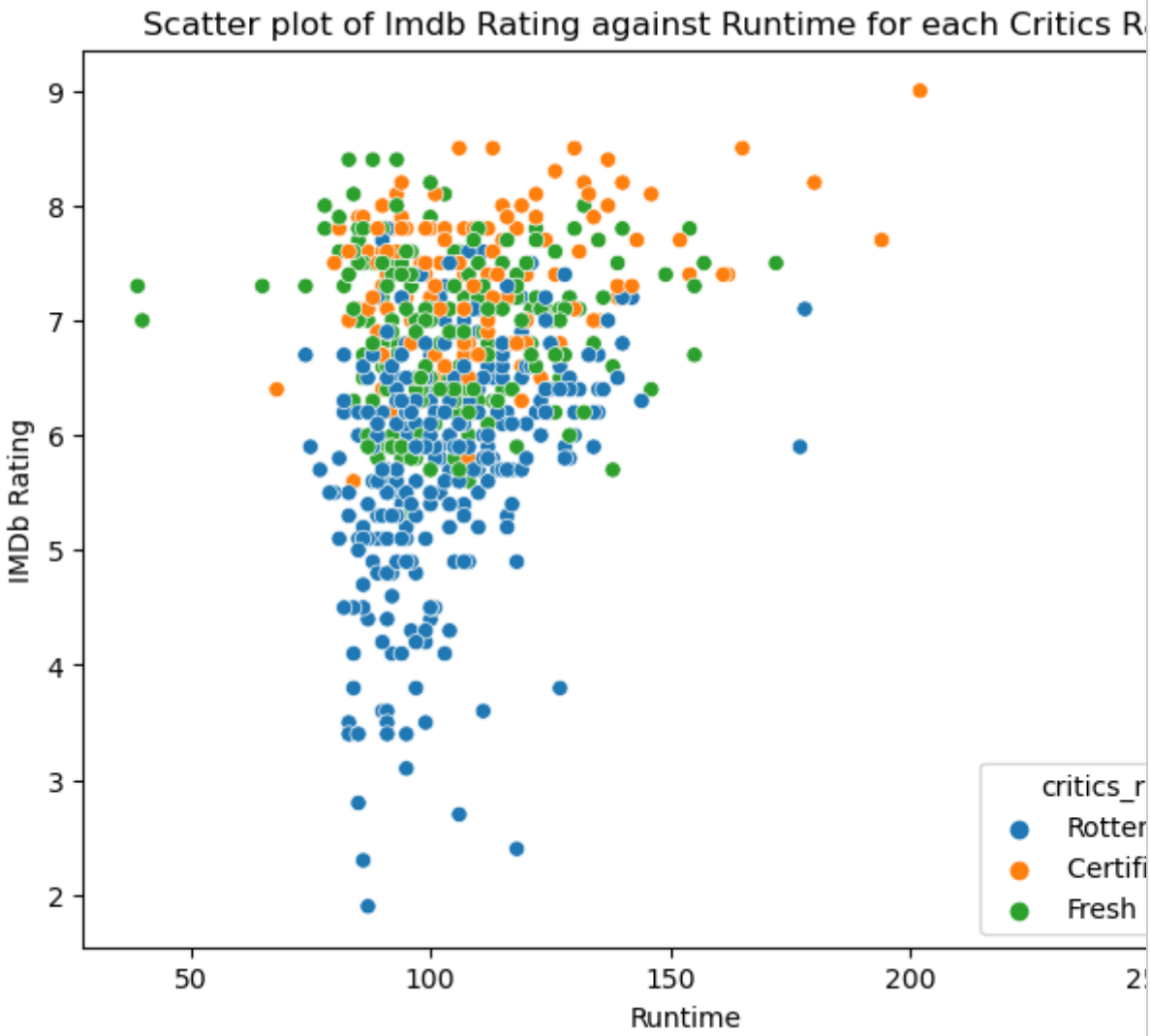


Q1(m) Visualization: Multiple variables

Create a small multiples (or faceted) scatter plot of `imdb_rating` (y-axis) against `runtime` (x-axis) for each `critics_rating`.

In [23]:

```
# Create a small multiples (or faceted) scatter plot of `imdb_rating` vs.
#   `runtime` for each `critics_rating`.
plt.figure(figsize = (8, 6))
sns.scatterplot(x = 'runtime',y = 'imdb_rating',hue = 'critics_rating',data = movies)

# Set labels and title
plt.xlabel('Runtime')
plt.ylabel('IMDb Rating')
plt.title('Scatter plot of Imdb Rating against Runtime for each Critics Rating')
plt.show()
```

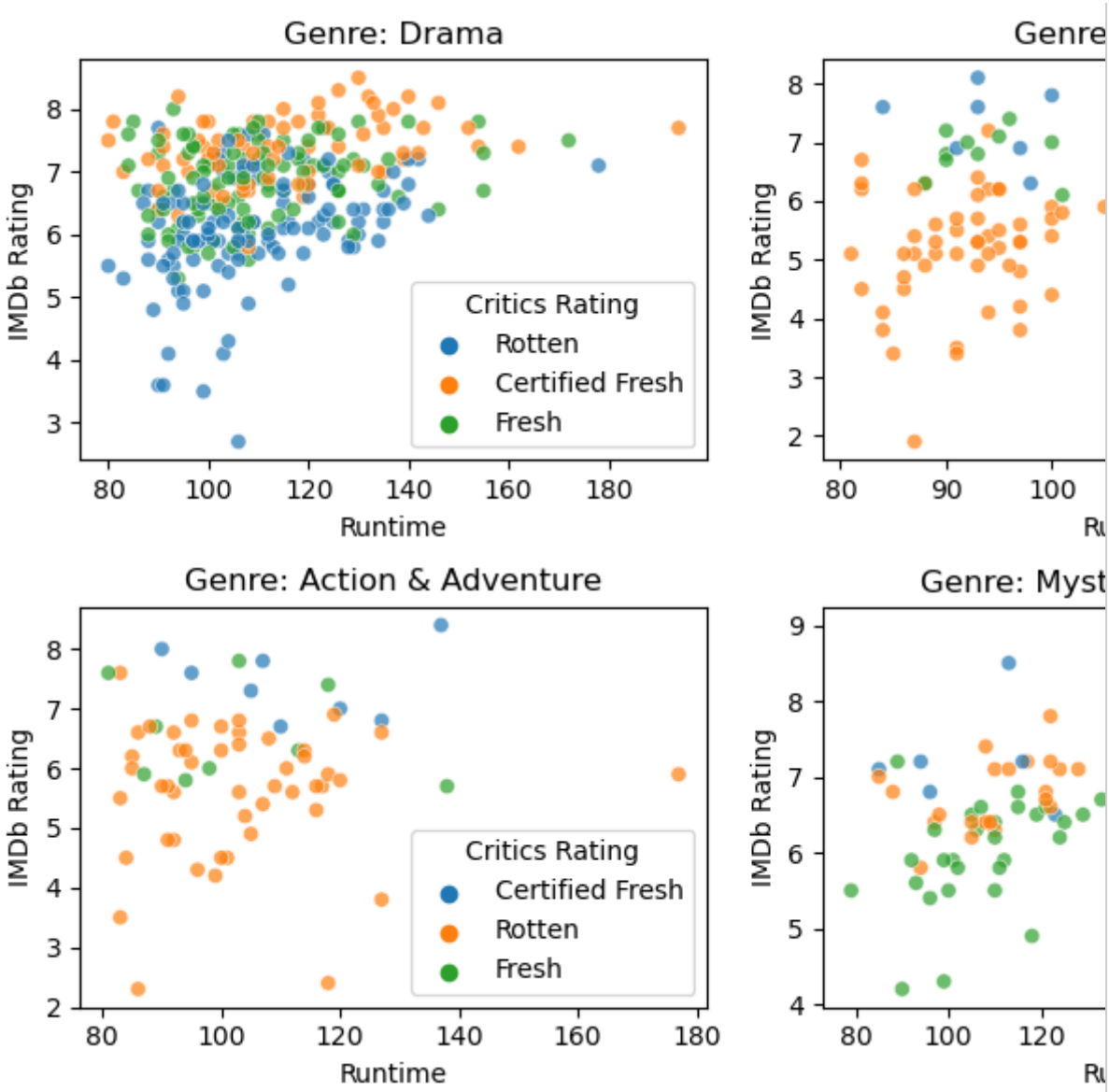


Q1(bonus)

Create a small multiples (or faceted) scatter plot of `imdb_rating` vs. `runtime` for each of the top 4 `genre` (in order of most frequent), colored with the `critics_rating`.

In [53]:

```
# Create a small multiples (or faceted) scatter plot of `imdb_rating` vs.
#   `runtime` for each of the top 4 `genre`, colored by the `critics_rating`

# Find the top 4 genres (most frequent)
top_genres = movies['genre'].value_counts().nlargest(4).index
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 6))

# Create scatter plots for each genre using a for loop
for i, genre in enumerate(top_genres):
    subset_df = movies[movies['genre'] == genre]
    row, col = divmod(i, 2)
    sns.scatterplot(x='runtime', y='imdb_rating', hue='critics_rating',
    data=subset_df, ax=axes[row, col], palette='tab10', alpha=0.7)
    axes[row, col].set_title(f'Genre: {genre}')
    axes[row, col].set_xlabel('Runtime')
    axes[row, col].set_ylabel('IMDb Rating')
    axes[row, col].legend(title='Critics Rating')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```



Q2 - Data Preprocessing

You are asked to write two functions `minmaxNorm` and `zscoreNorm` to perform normalization on passed in data. Do not just use preprocessing or normalization functions available in Python.

The function `minmaxNorm` should take four arguments

- `trData` - the training data (use to establish the data properties for normalization)
- `teData` - the testing data, (if supplied default: None), to also be normalized according the the same data properties
- `minV` - minimum value of new range (default: 0)
- `maxV` - maximum value of new range (default: 1)

The function `zscoreNorm` should take three arguments

- `trData` - the training data (use to establish the data properties for normalization)
- `teData` - the testing data, (if supplied default: None), to also be normalized according the the same data properties
- `madFlag` - boolean flag, if positive, use mean abs. deviation instead of standard deviation. (default: False)

For many of our tasks, the data is split into subsets (by sample) of training and testing data (see slides: 03.classify.part2). The training data is used to estimate the parameters needed in the transformation: for `zscoreNorm`, μ - mean and σ - standard deviation for each attribute; for `minmaxNorm`, *min* and *max* values for each attribute.

Both the training and testing data (if passed in) are to be transformed using the parameters from the training data and passed into the function. If the input is 2D a matrix/data frame/etc., then be sure to scale each feature (column) separately.

In Python, the functions should work on lists (a single variable dataset), np.arrays (1D - a single variable dataset and 2D), and data frames (see sample inputs below).

The functions should return a tuple-Python of the transformed training data and transformed testing data. In Python, each item of the tuple should be a numpy array. If no test data is included as input return None for this item of the tuple.

A few example test cases are supplied with more tested.

Hint: I suggest taking the input training/testing data and converting to the same type, e.g., numpy arrays. Then, do the normalization calculations on this single type of data object.

Test Examples

```
in1 = [0, 1, 2, 5]
out1 = minmaxNorm(in1, None)
print(out1)

in2 = np.array([0, 1, 2, 4, 8])
out2 = minmaxNorm(in2, None, -1, 1)
print(out2)

in3 = np.array([[1, 3, 4, 5], [2, 6, 5, 8], [3, 4, 6, 9]])
in3test = np.array([2.5, 5, 4, 7])
```

```
out3 = minmaxNorm(in3, in3test, -1, 1)
print(out3)

in4 = pd.DataFrame([[1, 3, 4, 5], [2, 6, 5, 8],
                    [3, 4, 6, 9]])
out4 = minmaxNorm(in4, in3test, 0, 10)
print(out4)
```

```
(array([0. , 0.2, 0.4, 1. ]), None)
(array([-1.  , -0.75, -0.5 ,  0.  ,  1.  ]), None)
(array([[-1.       , -1.       , -1.       , -1.       ],
       [ 0.       ,  1.       ,  0.       ,  0.5      ],
       [ 1.       , -0.33333333,  1.       ,  1.       ]]), array([ 0.5       ,  0.33333333, -1.        ,  0.        ]))
(array([[ 0.       ,  0.       ,  0.       ,  0.       ],
       [ 5.       , 10.       ,  5.       ,  7.5      ],
       [10.       ,  3.33333333, 10.       , 10.       ]]), array([7.5       , 6.66666667, 0.        , 5.        ]))
```

In [54]:

```
def minmaxNorm(trData, teData=None, minV=0, maxV=1):
    # Perform min-max normalization on trData and teData
    # return a tuple of the transformed data as numpy arrays

    # Change the training data to numpy arrays
    trData = np.array(trData)
    # Change the test data to numpy arrays if data is not None
    if teData is not None:
        teData = np.array(teData)

    # Calculate minimum values of training data features
    min_V = np.min(trData, axis  = 0)
    # Calculate maximum values of the training data features
    max_V = np.max(trData, axis = 0)

    # Perform minimum and maximum normalization
    trData_norm = (trData - min_V) / (max_V - min_V) * (maxV - minV) + minV
    # Perform minimum and maximum normalization if test data is not none
    if teData is not None:
        teData_norm = (teData - min_V) / (max_V - min_V) * (maxV - minV) + minV
    else:
        teData_norm = None

    return trData_norm, teData_norm


in1 = [0, 1, 2, 5]
out1 = minmaxNorm(in1, None)
print(out1)
```

```
in2 = np.array([0, 1, 2, 4, 8])
out2 = minmaxNorm(in2, None, -1, 1)
print(out2)

in3 = np.array([[1, 3, 4, 5], [2, 6, 5, 8], [3, 4, 6, 9]])
in3test = np.array([2.5, 5, 4, 7])
out3 = minmaxNorm(in3, in3test, -1, 1)
print(out3)

in4 = pd.DataFrame([[1, 3, 4, 5], [2, 6, 5, 8],
                    [3, 4, 6, 9]])
out4 = minmaxNorm(in4, in3test, 0, 10)
print(out4)
```

```
(array([0. , 0.2, 0.4, 1. ]), None)
(array([-1.  , -0.75, -0.5 ,  0.  ,  1.  ]), None)
(array([[-1.        , -1.        , -1.        , -1.        ],
       [ 0.        ,  1.        ,  0.        ,  0.5       ],
       [ 1.        , -0.33333333,  1.        ,  1.        ]]), array([ 0.5       ,  0.33333333, -1.        ,  0.
(array([[ 0.        ,  0.        ,  0.        ,  0.        ],
       [ 5.        , 10.        ,  5.        ,  7.5       ],
       [10.        ,  3.33333333, 10.        , 10.        ]]), array([7.5       , 6.66666667, 0.        , 5.
```

In [55]:

```
def zscoreNorm(trData, teData=None, madFlag=0):
    # Perform Gaussian normalization on trData and teData
    # return a tuple of the transformed data

     # Change the training data to numpy arrays
    trData = np.array(trData)
    # Change the test data to numpy arrays if data is not None
    if teData is not None:
        teData = np.array(teData)

    # Calculate mean and standard deviation training data features
    if madFlag:
        mean_V = np.mean(trData, axis = 0)
        std_V = np.mean(np.abs(trData - np.mean(trData, axis = 0)), axis = 0)
    else:
        mean_V = np.mean(trData, axis = 0)
        std_V = np.std(trData, axis = 0)

    # Perform z-score normalization
    trData_norm = (trData - mean_V) / std_V
    if teData is not None:
        teData_norm = (teData - mean_V) / std_V
```

```
    else:
        teData_norm = None

    return trData_norm, teData_norm
```

In [56]:
```
grader.check("q2")
```

Out [56]:
```
q2 results: All test cases passed!
```

Q3 - Data Preprocessing

Let's now compare the normalization functions that you wrote above to using the normalization functions that are available in standard libraries:

- Python: `MinMaxScaler` and `StandardScaler` or scale in `sklearn.preprocessing`

In [57]:
```
# train/test data
q3train = pd.DataFrame({'x1': [20, 37, 40, 60, 85, 120], 'x2': [-10, -8, 52, 3, 18, 23]})
q3test = pd.DataFrame({'x1': [42, 58, 101], 'x2': [-8, 42, 54]})

q3outA = minmaxNorm(q3train, None)
print(q3outA)

q3outB = zscoreNorm(q3train, q3test)
print(q3outB)

# use MinMaxScaler with default range parameters
scaler = MinMaxScaler()
q3outC = scaler.fit_transform(q3train)
print(q3outC)

# use StandardScaler
scaler = StandardScaler()
q3outDtrain = scaler.fit_transform(q3train)
q3outDtest = scaler.transform(q3test)
print(q3outDtrain)
print(q3outDtest)
```

```
(array([[0.        , 0.        ],
       [0.17      , 0.03225806],
       [0.2       , 1.        ],
       [0.4       , 0.20967742],
       [0.65      , 0.4516129 ],
       [1.        , 0.53225806]]), None)
```

```
(array([[-1.20221086, -1.08103207],
        [-0.69549389, -0.98702928],
        [-0.60607324,  1.83305438],
        [-0.00993563, -0.47001394],
        [ 0.7352364 ,  0.23500697],
        [ 1.77847723,  0.47001394]]), array([[-0.54645948, -0.98702928],
        [-0.06954939,  1.36304044],
        [ 1.21214649,  1.92705717]]))
[[0.         0.        ]
 [0.17       0.03225806]
 [0.2        1.        ]
 [0.4        0.20967742]
 [0.65       0.4516129 ]
 [1.         0.53225806]]
[[-1.20221086 -1.08103207]
 [-0.69549389 -0.98702928]
 [-0.60607324  1.83305438]
 [-0.00993563 -0.47001394]
 [ 0.7352364   0.23500697]
 [ 1.77847723  0.47001394]]
[[-0.54645948 -0.98702928]
 [-0.06954939  1.36304044]
 [ 1.21214649  1.92705717]]
```

In [58]:
```
grader.check("q3")
```

Out [58]: q3 results: All test cases passed!

Q4 - Performance Metrics

Write a function to calculate: (a) true positive rate, (b) false positive rate, (c) accuracy, and (d) Matthews Correlation Coefficient (MCC).

You can make use of `sklearn.metrics` functions.

The function will have inputs of `y_true` (np.array) - the true label for a set of samples and `y_pred` (np.array) - the predicted labels for a set of samples, and a threshold `thres_value` (float).

The function returns a list of the true positive rate, false positive rate, accuracy and MCC for the inputs where the predicted labels are thresholded at the provided value (using >= comparisons).

This function will then be used to create a DataFrames with rows corresponding with the 10 thresholds (y_pred values) and columns reporting the different thresholds, the true positive rate (TPR), false positive rate (FPR), accuracy (ACC), and Matthews correlation coefficient (MCC).

In [61]:

```
def calc_metrics(y_true, y_pred, thres_value):
    # Calculate tpr, fpr, accuracy, and MCC on input
    # Input:
    #  y_true - sample labels       (np.array)
    #  y_pred - sample predictions  (np.array)
    #  thres_value - threshold for predictions,  >=
    # Return list of tpr, fpr, accuracy, and MCC

    # Calculate the confusion matrix
    predictions_binary = (y_pred >= thres_value).astype(float)
    tn, fp, fn, tp = confusion_matrix(y_true, predictions_binary).ravel()
    # Calculate true positive rate
    tpr = tp / (tp + fn)
     # Calculate false positive rate
    fpr = fp / (fp + tn)
    # Calcuate accuracy
    acc = (tp + tn) / (tp + tn + fp + fn)
    # Calculate Matthews Correlation Coefficient (MCC)
    denominator = np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))
    mcc = (tp * tn - fp * fn) / denominator if denominator!= 0 else np.nan

    return tpr, fpr, acc, mcc



y_true = np.array([1,1,0,1,1,0,1,0,0,0])
y_pred = np.array([0.98,0.92,0.85,0.77,0.71,0.64,0.57,0.42,0.34,0.32])


perfDF = pd.DataFrame(columns = ['Threshold', 'TPR', 'FPR', 'ACC', 'MCC'])
i = 0
for thres in y_pred:
    tpr_val, fpr_val, acc_val, mcc_val = calc_metrics(y_true, y_pred, thres)
    perfDF.loc[i] = [thres, tpr_val, fpr_val, acc_val, mcc_val]
    i = i+1

perfDF
```

Out [61]:

	Threshold	TPR	FPR	ACC	MCC
0	0.98	0.2	0.0	0.6	0.333333

1	0.92	0.4	0.0	0.7	0.500000
2	0.85	0.4	0.2	0.6	0.218218
3	0.77	0.6	0.2	0.7	0.408248
4	0.71	0.8	0.2	0.8	0.600000
5	0.64	0.8	0.4	0.7	0.408248
6	0.57	1.0	0.4	0.8	0.654654
7	0.42	1.0	0.6	0.7	0.500000
8	0.34	1.0	0.8	0.6	0.333333
9	0.32	1.0	1.0	0.5	NaN

In [62]:
```
grader.check("q4")
```

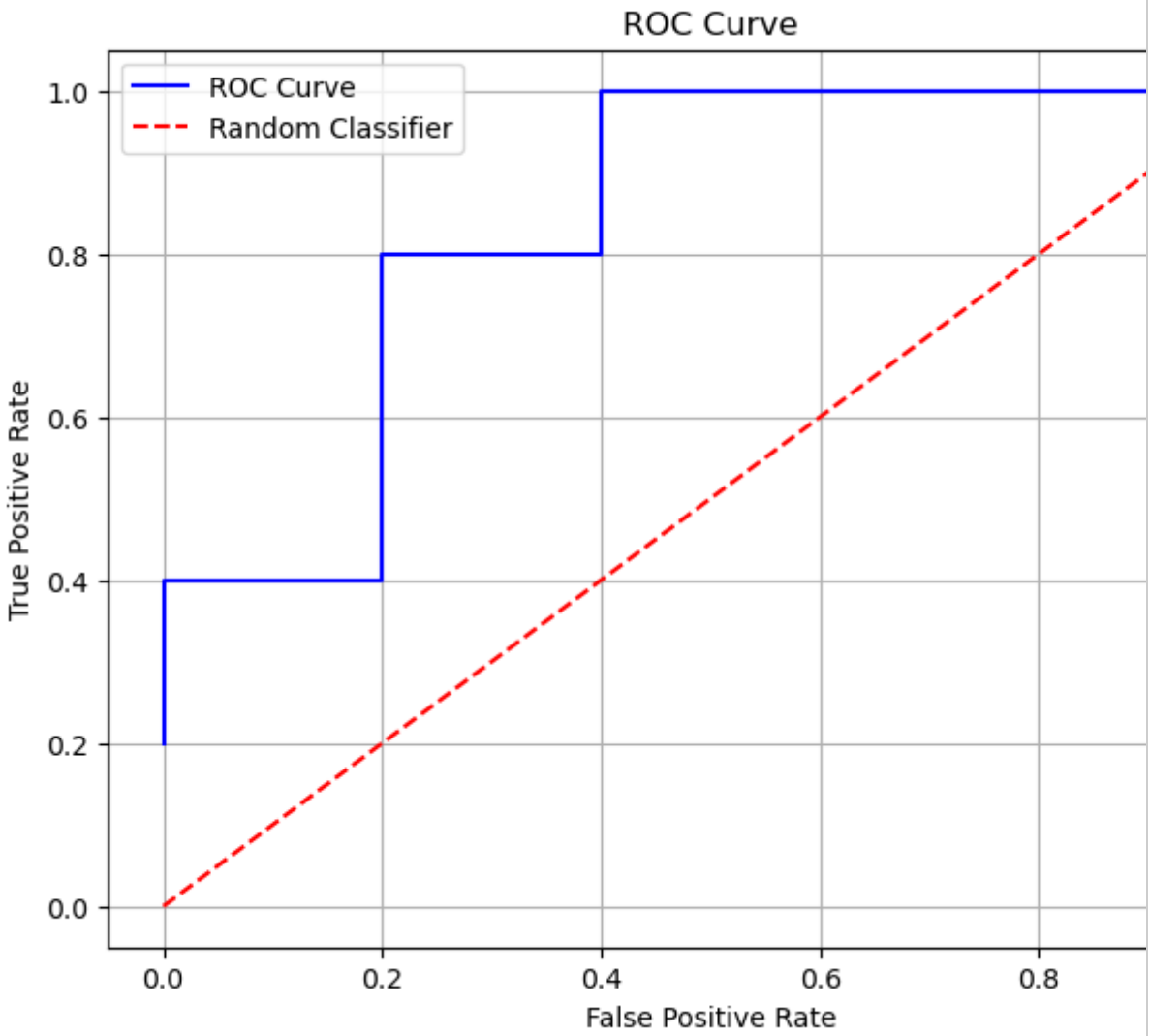
Out [62]: q4 results: All test cases passed!

Q5 - Plot ROC Curve:

Use the results from Question 4 to plot the ROC curve for the data.

Note, plot this curve using the standard plotting tools rather than any special library/package available for making ROC plots.

In [63]:
```
# Create a ROC curve using the results from Q4

# Get fpr and tpr values for plotting
fpr = perfDF['FPR'].values
tpr = perfDF['TPR'].values

plt.figure(figsize = (8,6))
plt.plot(fpr, tpr, label ='ROC Curve', color ='blue')
# Plot random classifier
plt.plot([0, 1], [0, 1], label = 'Random Classifier', linestyle = '--', color = 'red')

# Set labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')

plt.legend()
plt.grid()
plt.show()
```



Q6 - NBA Rookies

For this problem you will use a data set of rookie NBA players from 1980 - 2017 seasons. The dataset was collected from the NBA website API - https://www.nba.com.

You will use the data from their rookie year to predict whether a player will last at least 5 seasons in the leaque.

The data consists of variables:

- `PlayerID`, `Player` - variables to identify individual samples (ignore for prediction)
- `Tm`, `Year` - variables describing the year the player started and for what team (ignore for prediction)
- `TARGET` - This is the target / class feature to be predicted (whether the player was in the league for at least 5 years).

The remaining variables are predictor variables for the models. They come in pairs "*_DIFF" and "*_A" reporting the given statistic as the difference between Team A and Team B and the statistic itself for Team A.

- `Pos` - position of the player, power forward, point guard, shooting guard, center, etc.
- `Age` - player age
- `G` - sum of number of games played
- `GS` - sum of number of games started
- `MP` - sum of number of minutes played
- `PTS` - sum of number of points scored
- `FG` - sum of number of field goals made (both 2 and 3 pointers)
- `FGA` - sum of number of field goals attempted
- `FG%` - FG / FGA, percentage of field goals made
- `3P` - sum of the number of 3 pointers made
- `3PA` - sum of the number of 3 pointers attempted
- `3P%` - 3P / 3PA, percentage of 3 pointers made
- `2P` - sum of the number of 2 point shots made
- `2PA` - sum of the number of 2 point shots attempted
- `2P%` - 2P / 2PA, percentage of 2 point shots made
- `eFG%` - Effective Field Goal Percentage, (`FG` + 0.5 * `3P`)/`FGA`
- `FT` - sum of the number of free throws made
- `FTA` - sum of the number of free throws attempted
- `FT%` - FT / FTA, percentage of free throws made
- `ORB` - sum of the number of offensive rebounds
- `DRB` - sum of the number of defensive rebounds
- `TRB` - sum of the number of total rebounds
- `AST` - sum of the number of assists
- `STL` - sum of the number of steals
- `BLK` - sum of the number of blocks
- `TOV` - sum of the number of turnovers
- `PF` - sum of the number of personal fouls

More information on the stats used can be found:
https://www.nba.com/stats/help/glossary
Note, some of the abbreviations used here are slightly different

Q6(a) - Load Data

Load the `nba` data.

In [64]:
```
# Load nba data
nba = pd.read_csv('nba_rookies.csv')

nba.head()
```

Out [64]:

	PlayerID	Player	Pos	Age	Tm	Year	G	GS	MP	PTS	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	TARGET
0	abdelal01	Alaa Abdelnaby	PF	22	POR	1991	43	0.0	290	135	...	0.568182	27	62	89	12	4	12	22	39	True
1	abdulma02	Mahmoud Abdul-Rauf	PG	21	DEN	1991	67	19.0	1505	942	...	0.857143	34	87	121	206	55	4	110	149	True
2	abdulta01	Tariq Abdul-Wahad	SG	23	SAC	1998	59	16.0	959	376	...	0.672000	44	72	116	51	35	13	65	81	True
3	abdursh01	Shareef Abdur-Rahim	PF	20	VAN	1997	80	71.0	2802	1494	...	0.745665	216	339	555	175	79	79	225	199	True
4	abrinal01	Álex Abrines	SG	23	OKC	2017	68	6.0	1055	406	...	0.897959	18	68	86	40	37	8	33	114	False

[5 rows x 32 columns]

In [65]:
```
grader.check("q6a")
```

Out [65]:
q6a results: All test cases passed!

Q6(b) - Missing Data

Let's investigate any missing values in the nba data.

First, calculate and report the percentage of missing data for that variable (percentage of rows) in a DataSeries, `miss_nba`.

In [66]:
```
# Calculate the percentage of missing values and report
miss_nba = (nba.isnull().sum()/len(nba)) * 100

miss_nba
```

Out [66]:
```
PlayerID     0.000000
Player       0.000000
Pos          0.000000
Age          0.000000
Tm           0.000000
Year         0.000000
G            0.000000
GS           4.807325
MP           0.000000
PTS          0.000000
FG           0.000000
FGA          0.000000
```

```
FG%          0.763068
3P           0.000000
3PA          0.000000
3P%         23.693247
2P           0.000000
2PA          0.000000
2P%          0.915681
eFG%         0.763068
FT           0.000000
FTA          0.000000
FT%          6.676841
ORB          0.000000
DRB          0.000000
TRB          0.000000
AST          0.000000
STL          0.000000
BLK          0.000000
TOV          0.000000
PF           0.000000
TARGET       0.000000
dtype: float64
```

In [67]:
```
grader.check("q6b")
```

Out [67]: q6b results: All test cases passed!

Explore where these missing values you in the following code cell.

BE SURE TO COMMENT OUT YOUR CODE BEFORE SUBMITTING

In [68]:
```
# explore where the missing values are in your data.

# The missing values are in the following columns:
# GS- games started
# 3P / 3PA, percentage of 3 pointers made(3P), and
# FT / FTA, percentage of free throws made(FT%)

#to explore the missing values
#nba.GS.isnull()
#nba['3P%'].isnull()
#nba['FT%'].isnull()
```

Q6(c) - Handle missing data

By investigating where the missing data is, I hope you discovered the following:

The missing values reside in two main types of columns (and for two different reasons):

- `GS` the game started column has missing values for almost all rookies in the 1980 and 1981 seasons with a few exceptions, e.g., Larry Bird, Clint Richardson, etc.
- Columns that calculate a percentage, `FG%`, `3P%`, `2P%`, `eFG%`, `FT%` Here the missing values are due to a divide by zero.

Due to the different reasons for the data missing values will be handled differently in each situation.

First, the missing values for `GS`, because there is not way to impute these values (and there are so few players in the 1980 and 1981 season having this information), the **all** samples from these two seasons should be deleted.

For the missing values in the "percentage" columns, replace those missing values with 0.

Call you new DataFrame after performing these operations `nba2`.

Note There may be situations where how you handle missing data (in particular, using imputation methods) should be done in the workflow/pipeline after already splitting for training/testing data in order to avoid possible data leakage. However, the methods we employ here for dealing the the missing data are not using global properties to perform the imputation, therefore can be done at this state of the analysis.

In [69]:

```
# Copy the nba dataframe to a new dataframe(nba2)
nba2 = nba.copy()
# Drop rows from the 1980 and 1981 seasons
nba2 = nba2[(nba2['Year'] != 1980) & (nba2['Year'] != 1981)]
# Drop missing values with in GS
nba2 = nba2.dropna(subset = ['GS'])
# Replace missing values with 0 in specified columns
columns_to_replace = ['FG%', '3P%', '2P%', 'eFG%', 'FT%']
nba2[columns_to_replace] = nba2[columns_to_replace].fillna(0)
nba2.shape
```

Out [69]:

```
(2487, 32)
```

In [70]:
```
grader.check("q6c")
```

Out [70]:
```
q6c results: All test cases passed!
```

Q6(d) - Labels

Let's understand the what we should expect as a baseline performance for predicting whether the rookie play remains in the league for at least 5 years.

(i). What fraction of players have a positive target label (in the league for at least 5 years)? Value should be in between 0 and 1.
(ii). What should a constant classifier model predict? A *constant classifier* always predicts the same value no matter the input.
(iii). What is the error rate of the constant classifier? Value should be in between 0 and 1.

Answer the following questions below. Note, you should not use any `sklearn` functions, but simply look at properties of the data labels.

In [73]:
```
# find the number of players with positive target label
postive_tar = (nba2['TARGET'] == True).sum()
# find the total number of target labels
total_tar = nba2['TARGET'].count()
# fraction of players that has positive target label
q6d_i = postive_tar/total_tar

# a constant classifier will predict the majority's label
# find the number of players with negative target label
negative_tar = (nba2['TARGET'] == False).sum()
neg_frac = negative_tar/total_tar
#print true if positive fraction is larger than negative fraction
q6d_ii = np.where(q6d_i > neg_frac, 'True', 'False')

q6d_iii = 1- q6d_i
```

In [74]:
```
grader.check("q6d")
```

Out [74]:
```
q6d results: All test cases passed!
```

Q6(e) - Prepare the data

At this point, we need to set up the data in order to be used in the classification models.

We want to create a DataFrame `nbaX` for the predictor variables and `nbaY` for the target variable.

The target variable, `nbaY` will just be the `Target` column of the `nba2` data set.

The predictor variables have more considerations.

We want to exclude player information such as IDs, `PlayerID` and names `Player`, that are identifying and not predictive.

You should also exclude these other factors (note, that some of these variables may in fact be predictive but we are going to exclude at this time):

- `POS` - player position
- `Tm` - team
- `Year` - rookie year

Finally, several of the numeric predictive variables are not only related, but can be directly calculated from one another, e.g., `FG%` = `FG` / `FGA`. Having variables that are closely related, or in this case redundant may actually hinder the predictive models.

Therefore, exclude the following variables: `FG`, `3P`, `2P`, `eFG%`, `FT`, `TRB`.

In [77]:
```
# Create the target variable
nbaY = nba2['TARGET']
# Create a variable for columns to be excluded
columns_to_exclude = ['PlayerID',
'Player','Pos','Tm','Year','FG','3P','2P','eFG%','FT','TRB','TARGET','Age']
# Create a new dataframe by dropping columns to be excluded
nbaX = nba2.drop(columns = columns_to_exclude)

print(nbaX.shape)
print(nbaY.shape)
```

```
(2487, 19)
(2487,)
```

In [78]:
```
grader.check("q6e")
```

Out [78]: q6e results: All test cases passed!

Q6(f) - Model Selection and Evaluation: Three-fold Split

Split the data into training, validation and test sets with 60, 20, and 20% of the data respectively. Make sure to split the data such that the distribution of class labels is approximately equal across splits - “stratify”.

Set the seed for the random generator in `random_state` to ”4821”

In [79]:
```
# Split of the test set
X_trainval, X_test, y_trainval, y_test = train_test_split(nbaX, nbaY, test_size = 0.2,
stratify = nbaY, random_state = 4821)

# Split trainval into train + val
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, test_size = 0.25,
stratify = y_trainval, random_state = 4821)
```

In [80]:
```
grader.check("q6f")
```

Out [80]: q6f results: All test cases passed!

Q6(g) - Scaling

Scale the predictor data with standard scaling (Gaussian normalizaiton).

Make sure to use training data set to set scaling parameters and apply those parameters to scaling the training and validation data.

Use the train+val to scale the train+val data, and use those parameters to scale the test data to evaluate the best model.

Helpful functions: Python - `StandardScaler` from `sklearn.preprocessing`.

In [81]:
```
# Scale the predictor data
scaler =  StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_val_sc = scaler.transform(X_val)
```

```
scaler_final = StandardScaler()
X_trainval_sc = scaler_final.fit_transform(X_trainval)
X_test_sc = scaler_final.transform(X_test)
```

In [82]:
```
grader.check("q6g")
```

Out [82]: q6g results: All test cases passed!

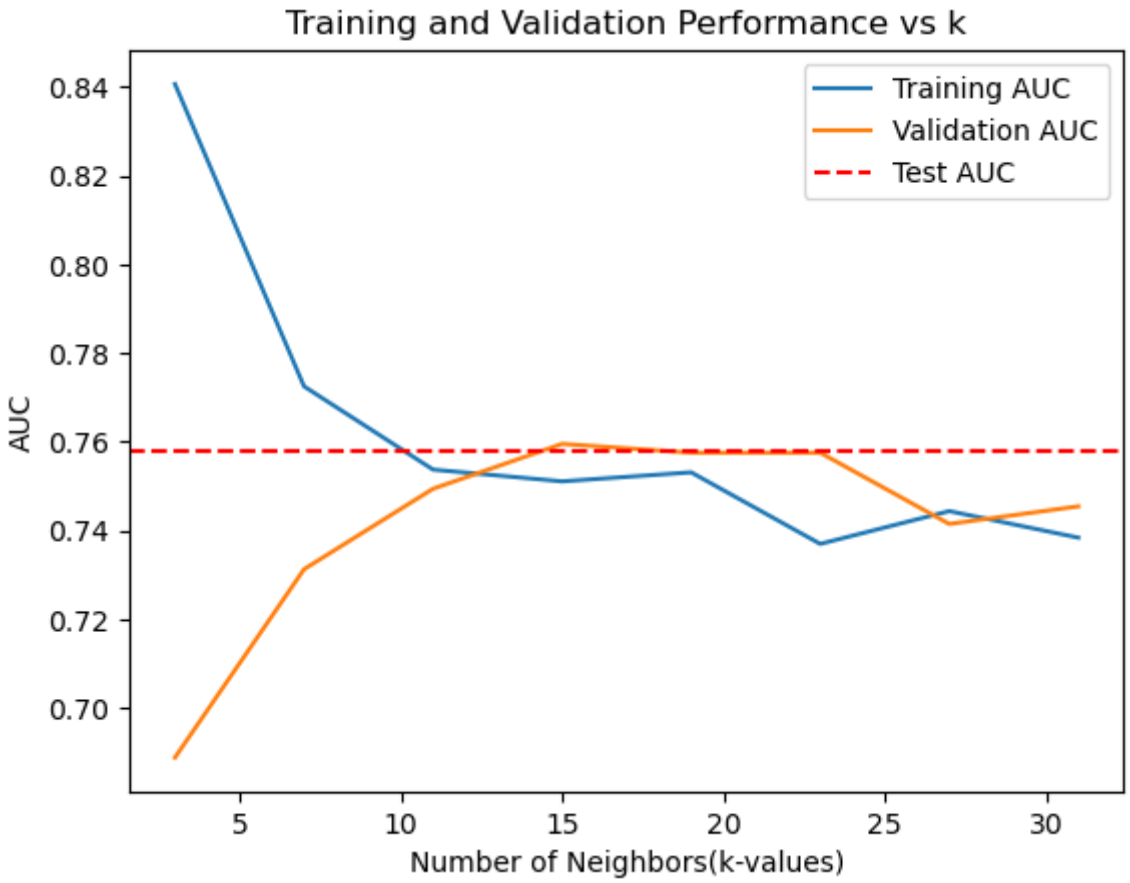
Q6(h) - KNN - K Nearest Neighbors

For values of k, 3, 7, 11, 15, 19, 23, 27, 31, find the best k-nearest-neighbor classifier using the three-fold split of data.

- fit a k nearest neighbors model to the training data for each value of k
- evaluate the classifier on the training and validation set using AUC
- select the best value of k
- create a model on train+validation with the *best* value of k
- evaluate the classifier on the test data.
- plot the training and validation performance vs k;
  add a line showing the test performance

In [93]:
```
# Create a variable for values of k
kvals = [3, 7, 11, 15, 19, 23, 27, 31]

# Create arrays to store AUC score
knn_auc_val = np.zeros((1,len(kvals)))
knn_auc_tr = np.zeros((1,len(kvals)))


# Fit a k nearest neighbors model to the training data
# Evaluate the classifier on the training and validation set using AUC
for n,k in enumerate(kvals):
    knn = KNeighborsClassifier(n_neighbors =  k)
    knn.fit(X_train_sc, y_train)

    y_train_pred = knn.predict(X_train_sc)
    y_val_pred = knn.predict(X_val_sc)

    knn_auc_tr[0,n] = roc_auc_score(y_train,y_train_pred)
    knn_auc_val[0,n] = roc_auc_score(y_val, y_val_pred)

# Select the best value of k
knn_index = np.argmax(knn_auc_val)
knn_bestk = kvals[knn_index]
```

```
# Create a best model on train+validation
knn_f = KNeighborsClassifier(n_neighbors = knn_bestk)
knn_f.fit(X_trainval_sc,y_trainval)
y_pred_val = knn_f.predict(X_test_sc)

# Evaluate the classifier on the test data.
knn_auc_test = roc_auc_score(y_test,y_pred_val)


# Plot the training and validation performance vs k.
plt.plot(kvals,knn_auc_tr.T, label='Training AUC')
plt.plot(kvals,knn_auc_val.T, label='Validation AUC')
# Add a line for test performance
plt.axhline(y=knn_auc_test, linestyle='--', label='Test AUC', color='r')

# Set labels and title
plt.ylabel('AUC')
plt.xlabel('Number of Neighbors(k-values)')
plt.title('Training and Validation Performance vs k')
plt.legend()
plt.show()

print('Best k:  %d' % (knn_bestk))
print('Test Perf:  %.6f' % (knn_auc_test))
```



Best k: 15
Test Perf: 0.757791

In [155]:
```
grader.check("q6h")
```

Out [155]: q6h results: All test cases passed!

Q6(i) - Decision Trees

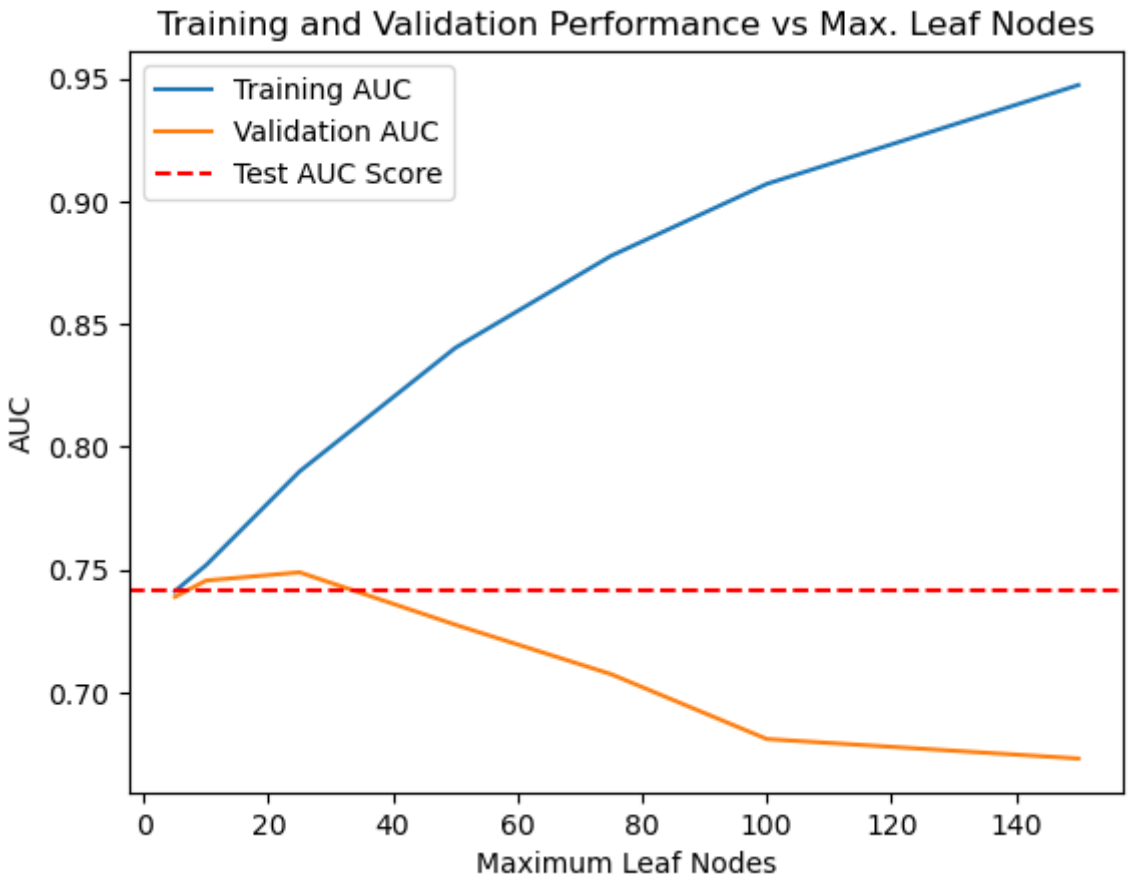
For values of max_leaf_nodes nodes 5, 10, 25, 50, 75, 100, fit the Decision Trees classifier to the training data.

- fit a decision tree model to the training data (use `random_state=4821`) for each of the max_leaf_nodes values
- evaluate the classifier on the training and validation set using AUC
- select the best max_leaf_nodes
- retrain the a model on train+validation with the *best* value of max_leaf_nodes
- report the auc on the testing data.
- plot the train and validation AUC vs max_leaf_nodes;
  add a line for the test AUC performance
- print out the best tree.

In [100]:

```
# Create a variable for maximum leaf nodes
nodes = [5, 10, 25, 50, 75, 100, 150]
# Create arrays to store AUC score
dt_auc_val = np.zeros((1,len(nodes)))
dt_auc_tr = np.zeros((1,len(nodes)))

# Fit a decision tree model to the training data (use random_state=4821)
# Evaluate the classifier on the training and validation set using auc
for n, max_leaf_nodes in enumerate(nodes):

    dt = DecisionTreeClassifier(max_leaf_nodes = max_leaf_nodes, random_state =
4821)
    dt.fit(X_train_sc, y_train)

    y_train_pred = dt.predict(X_train_sc)
    y_val_pred = dt.predict(X_val_sc)

    dt_auc_tr[0,n] = roc_auc_score(y_train, y_train_pred)
    dt_auc_val[0,n] = roc_auc_score(y_val, y_val_pred)

# Select the best max_leaf_nodes
dt_bestn = nodes[np.argmax(dt_auc_val)]

# Evaluate the classifier on the test data.
dt_auc_test = roc_auc_score(y_test,y_pred_val)

# Retrain the best model on train+validation
# Report the AUC on the testing data.
dt_f = DecisionTreeClassifier(max_leaf_nodes = dt_bestn, random_state = 4821)
dt_f.fit(X_trainval_sc, y_trainval)
y_pred_val = dt_f.predict(X_test_sc)
dt_auc_test = roc_auc_score(y_test, y_pred_val)

# Plot the train and validation auc vs max_leaf_nodes.
plt.plot(nodes, dt_auc_tr.T, label='Training AUC')
plt.plot(nodes, dt_auc_val.T,label='Validation AUC')

# Add a line for test performance
plt.axhline(y=dt_auc_test, linestyle='--', label='Test AUC Score', color='r')

# Set labels and title
plt.xlabel('Maximum Leaf Nodes')
plt.ylabel('AUC')
plt.title('Training and Validation Performance vs Max. Leaf Nodes')
plt.legend()
plt.show()
```

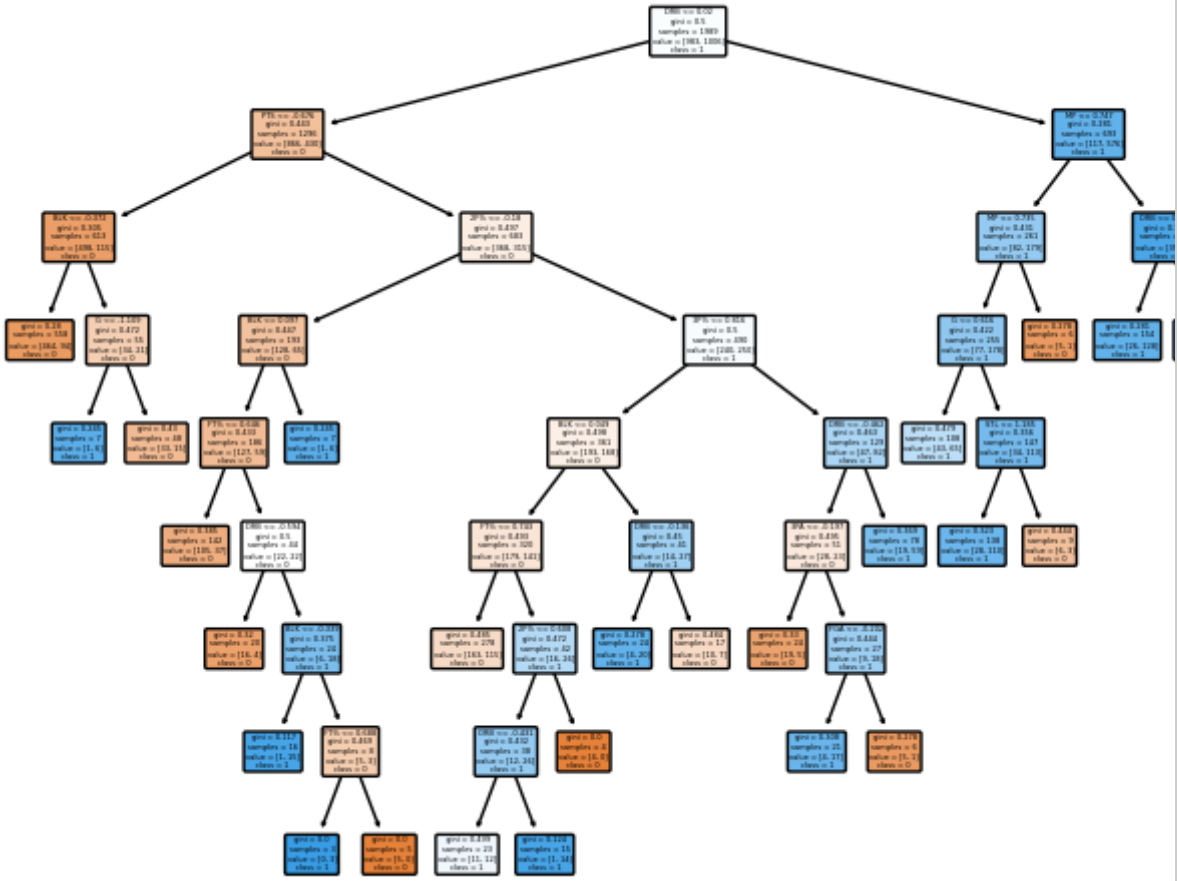
```
print('Best max_leaf_nodes:  %d' % (dt_bestn))
print('Test Perf:  %.6f' % (dt_auc_test))
```




```
Best max_leaf_nodes:  25
Test Perf:  0.741870
```

In [99]:

```
from sklearn.tree import plot_tree
# print out the tree
feature_names_list = X_train.columns.tolist()
plt.figure(figsize = (8, 6))
columns = [X_train.columns]
plot_tree(dt_f, filled = True, feature_names = feature_names_list, class_names = ["0",
"1"], rounded = True)
plt.show()
```



In [167]:
```
grader.check("q6i")
```

Out [167]: q6i results: All test cases passed!

Q6(j) - Naive Bayes

Train a Gaussian Naive Bayes model on training + validation data and report the training+val and testing data performance (auc).

In [168]:
```
# q6j
# Retrain the best model on train+validation
# Report the AUC on the testing data
nb = naive_bayes.GaussianNB()
nb.fit(X_trainval_sc, y_trainval)
nb_auc_trainval =  roc_auc_score(y_trainval, nb.predict(X_trainval_sc))
nb_auc_test = roc_auc_score(y_test, nb.predict(X_test_sc))
```

In [169]:
```
grader.check("q6j")
```

Out [169]:

```
q6j results: All test cases passed!
```

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

NOTE the submission must be run on the campus linux machines. See the instruction in the Canvas assignment.

In [56]:

```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False, run_tests=True)
```

```
Running your submission against local test cases...


Your submission received the following results when run against available test cases:


<IPython.core.display.HTML object>
```

▾ **.OTTER_LOG** Download

1	Large file hidden. You can download it using the button above.

▾ **__zip_filename__** Download

1	a2-5831_2024_02_21T21_45_08_371736.zip