{

"cells": [

{

"cell\_type": "code",

"execution\_count": 1,

"metadata": {},

"outputs": [],

"source": [

"# Filtering out the warnings\n",

"\n",

"import warnings\n",

"\n",

"warnings.filterwarnings('ignore')"

]

},

{

"cell\_type": "code",

"execution\_count": 2,

"metadata": {},

"outputs": [],

"source": [

"# Importing the required libraries\n",

"\n",

"import numpy as np\n",

"import pandas as pd\n",

"import seaborn as sns\n",

"import matplotlib.pyplot as plt\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# <font color = blue> IMDb Movie Assignment </font>\n",

"\n",

"You have the data for the 100 top-rated movies from the past decade along with various pieces of information about the movie, its actors, and the voters who have rated these movies online. In this assignment, you will try to find some interesting insights into these movies and their voters, using Python."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Task 1: Reading the data"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 1.1: Read the Movies Data.\n",

"\n",

"Read the movies data file provided and store it in a dataframe `movies`."

]

},

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {},

"outputs": [],

"source": [

"# Read the csv file using 'read\_csv'. Please write your dataset location here.\n",

"movies=pd.read\_csv(\"movie+assignment+Data.csv\")\n",

"# 100 rows and 62 columns"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 1.2: Inspect the Dataframe\n",

"\n",

"Inspect the dataframe for dimensions, null-values, and summary of different numeric columns."

]

},

{

"cell\_type": "code",

"execution\_count": 4,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(100, 62)"

]

},

"execution\_count": 4,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# Check the number of rows and columns in the dataframe\n",

"movies.shape\n"

]

},

{

"cell\_type": "code",

"execution\_count": 5,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"<class 'pandas.core.frame.DataFrame'>\n",

"RangeIndex: 100 entries, 0 to 99\n",

"Data columns (total 62 columns):\n",

" # Column Non-Null Count Dtype \n",

"--- ------ -------------- ----- \n",

" 0 Title 100 non-null object \n",

" 1 title\_year 100 non-null int64 \n",

" 2 budget 100 non-null int64 \n",

" 3 Gross 100 non-null int64 \n",

" 4 actor\_1\_name 100 non-null object \n",

" 5 actor\_2\_name 100 non-null object \n",

" 6 actor\_3\_name 100 non-null object \n",

" 7 actor\_1\_facebook\_likes 100 non-null int64 \n",

" 8 actor\_2\_facebook\_likes 99 non-null float64\n",

" 9 actor\_3\_facebook\_likes 98 non-null float64\n",

" 10 IMDb\_rating 100 non-null float64\n",

" 11 genre\_1 100 non-null object \n",

" 12 genre\_2 97 non-null object \n",

" 13 genre\_3 74 non-null object \n",

" 14 MetaCritic 95 non-null float64\n",

" 15 Runtime 100 non-null int64 \n",

" 16 CVotes10 100 non-null int64 \n",

" 17 CVotes09 100 non-null int64 \n",

" 18 CVotes08 100 non-null int64 \n",

" 19 CVotes07 100 non-null int64 \n",

" 20 CVotes06 100 non-null int64 \n",

" 21 CVotes05 100 non-null int64 \n",

" 22 CVotes04 100 non-null int64 \n",

" 23 CVotes03 100 non-null int64 \n",

" 24 CVotes02 100 non-null int64 \n",

" 25 CVotes01 100 non-null int64 \n",

" 26 CVotesMale 100 non-null int64 \n",

" 27 CVotesFemale 100 non-null int64 \n",

" 28 CVotesU18 100 non-null int64 \n",

" 29 CVotesU18M 100 non-null int64 \n",

" 30 CVotesU18F 100 non-null int64 \n",

" 31 CVotes1829 100 non-null int64 \n",

" 32 CVotes1829M 100 non-null int64 \n",

" 33 CVotes1829F 100 non-null int64 \n",

" 34 CVotes3044 100 non-null int64 \n",

" 35 CVotes3044M 100 non-null int64 \n",

" 36 CVotes3044F 100 non-null int64 \n",

" 37 CVotes45A 100 non-null int64 \n",

" 38 CVotes45AM 100 non-null int64 \n",

" 39 CVotes45AF 100 non-null int64 \n",

" 40 CVotes1000 100 non-null int64 \n",

" 41 CVotesUS 100 non-null int64 \n",

" 42 CVotesnUS 100 non-null int64 \n",

" 43 VotesM 100 non-null float64\n",

" 44 VotesF 100 non-null float64\n",

" 45 VotesU18 100 non-null float64\n",

" 46 VotesU18M 100 non-null float64\n",

" 47 VotesU18F 100 non-null float64\n",

" 48 Votes1829 100 non-null float64\n",

" 49 Votes1829M 100 non-null float64\n",

" 50 Votes1829F 100 non-null float64\n",

" 51 Votes3044 100 non-null float64\n",

" 52 Votes3044M 100 non-null float64\n",

" 53 Votes3044F 100 non-null float64\n",

" 54 Votes45A 100 non-null float64\n",

" 55 Votes45AM 100 non-null float64\n",

" 56 Votes45AF 100 non-null float64\n",

" 57 Votes1000 100 non-null float64\n",

" 58 VotesUS 100 non-null float64\n",

" 59 VotesnUS 100 non-null float64\n",

" 60 content\_rating 100 non-null object \n",

" 61 Country 100 non-null object \n",

"dtypes: float64(21), int64(32), object(9)\n",

"memory usage: 48.6+ KB\n"

]

}

],

"source": [

"# Check the column-wise info of the dataframe\n",

"movies.info()\n"

]

},

{

"cell\_type": "code",

"execution\_count": 6,

"metadata": {},

"outputs": [

{

"data": {

"text/html": [

"<div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>title\_year</th>\n",

" <th>budget</th>\n",

" <th>Gross</th>\n",

" <th>actor\_1\_facebook\_likes</th>\n",

" <th>actor\_2\_facebook\_likes</th>\n",

" <th>actor\_3\_facebook\_likes</th>\n",

" <th>IMDb\_rating</th>\n",

" <th>MetaCritic</th>\n",

" <th>Runtime</th>\n",

" <th>CVotes10</th>\n",

" <th>...</th>\n",

" <th>Votes1829F</th>\n",

" <th>Votes3044</th>\n",

" <th>Votes3044M</th>\n",

" <th>Votes3044F</th>\n",

" <th>Votes45A</th>\n",

" <th>Votes45AM</th>\n",

" <th>Votes45AF</th>\n",

" <th>Votes1000</th>\n",

" <th>VotesUS</th>\n",

" <th>VotesnUS</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>count</th>\n",

" <td>100.000000</td>\n",

" <td>1.000000e+02</td>\n",

" <td>1.000000e+02</td>\n",

" <td>100.000000</td>\n",

" <td>99.000000</td>\n",

" <td>98.000000</td>\n",

" <td>100.000000</td>\n",

" <td>95.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>...</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.00000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" <td>100.000000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>mean</th>\n",

" <td>2012.820000</td>\n",

" <td>7.838400e+07</td>\n",

" <td>1.468679e+08</td>\n",

" <td>13407.270000</td>\n",

" <td>7377.303030</td>\n",

" <td>3002.153061</td>\n",

" <td>7.883000</td>\n",

" <td>78.252632</td>\n",

" <td>126.420000</td>\n",

" <td>73212.160000</td>\n",

" <td>...</td>\n",

" <td>7.982000</td>\n",

" <td>7.732000</td>\n",

" <td>7.723000</td>\n",

" <td>7.780000</td>\n",

" <td>7.65100</td>\n",

" <td>7.624000</td>\n",

" <td>7.770000</td>\n",

" <td>7.274000</td>\n",

" <td>7.958000</td>\n",

" <td>7.793000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>std</th>\n",

" <td>1.919491</td>\n",

" <td>7.445295e+07</td>\n",

" <td>1.454004e+08</td>\n",

" <td>10649.037862</td>\n",

" <td>13471.568216</td>\n",

" <td>6940.301133</td>\n",

" <td>0.247433</td>\n",

" <td>9.122066</td>\n",

" <td>19.050799</td>\n",

" <td>82669.594746</td>\n",

" <td>...</td>\n",

" <td>0.321417</td>\n",

" <td>0.251814</td>\n",

" <td>0.260479</td>\n",

" <td>0.282128</td>\n",

" <td>0.21485</td>\n",

" <td>0.213258</td>\n",

" <td>0.301344</td>\n",

" <td>0.361987</td>\n",

" <td>0.232327</td>\n",

" <td>0.264099</td>\n",

" </tr>\n",

" <tr>\n",

" <th>min</th>\n",

" <td>2010.000000</td>\n",

" <td>3.000000e+06</td>\n",

" <td>2.238380e+05</td>\n",

" <td>39.000000</td>\n",

" <td>12.000000</td>\n",

" <td>0.000000</td>\n",

" <td>7.500000</td>\n",

" <td>62.000000</td>\n",

" <td>91.000000</td>\n",

" <td>6420.000000</td>\n",

" <td>...</td>\n",

" <td>7.300000</td>\n",

" <td>7.300000</td>\n",

" <td>7.200000</td>\n",

" <td>7.200000</td>\n",

" <td>7.10000</td>\n",

" <td>7.100000</td>\n",

" <td>7.000000</td>\n",

" <td>6.400000</td>\n",

" <td>7.500000</td>\n",

" <td>7.300000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>25%</th>\n",

" <td>2011.000000</td>\n",

" <td>1.575000e+07</td>\n",

" <td>4.199752e+07</td>\n",

" <td>1000.000000</td>\n",

" <td>580.000000</td>\n",

" <td>319.750000</td>\n",

" <td>7.700000</td>\n",

" <td>72.000000</td>\n",

" <td>114.750000</td>\n",

" <td>30587.000000</td>\n",

" <td>...</td>\n",

" <td>7.700000</td>\n",

" <td>7.600000</td>\n",

" <td>7.500000</td>\n",

" <td>7.600000</td>\n",

" <td>7.50000</td>\n",

" <td>7.475000</td>\n",

" <td>7.500000</td>\n",

" <td>7.100000</td>\n",

" <td>7.800000</td>\n",

" <td>7.600000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>50%</th>\n",

" <td>2013.000000</td>\n",

" <td>4.225000e+07</td>\n",

" <td>1.070266e+08</td>\n",

" <td>13000.000000</td>\n",

" <td>1000.000000</td>\n",

" <td>626.500000</td>\n",

" <td>7.800000</td>\n",

" <td>78.000000</td>\n",

" <td>124.000000</td>\n",

" <td>54900.500000</td>\n",

" <td>...</td>\n",

" <td>8.000000</td>\n",

" <td>7.700000</td>\n",

" <td>7.700000</td>\n",

" <td>7.800000</td>\n",

" <td>7.65000</td>\n",

" <td>7.600000</td>\n",

" <td>7.800000</td>\n",

" <td>7.300000</td>\n",

" <td>7.950000</td>\n",

" <td>7.750000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>75%</th>\n",

" <td>2014.000000</td>\n",

" <td>1.500000e+08</td>\n",

" <td>2.107548e+08</td>\n",

" <td>20000.000000</td>\n",

" <td>11000.000000</td>\n",

" <td>1000.000000</td>\n",

" <td>8.100000</td>\n",

" <td>83.500000</td>\n",

" <td>136.250000</td>\n",

" <td>80639.000000</td>\n",

" <td>...</td>\n",

" <td>8.200000</td>\n",

" <td>7.900000</td>\n",

" <td>7.900000</td>\n",

" <td>8.000000</td>\n",

" <td>7.80000</td>\n",

" <td>7.800000</td>\n",

" <td>7.925000</td>\n",

" <td>7.500000</td>\n",

" <td>8.100000</td>\n",

" <td>7.925000</td>\n",

" </tr>\n",

" <tr>\n",

" <th>max</th>\n",

" <td>2016.000000</td>\n",

" <td>2.600000e+08</td>\n",

" <td>9.366622e+08</td>\n",

" <td>35000.000000</td>\n",

" <td>96000.000000</td>\n",

" <td>46000.000000</td>\n",

" <td>8.800000</td>\n",

" <td>100.000000</td>\n",

" <td>180.000000</td>\n",

" <td>584839.000000</td>\n",

" <td>...</td>\n",

" <td>8.800000</td>\n",

" <td>8.700000</td>\n",

" <td>8.700000</td>\n",

" <td>8.500000</td>\n",

" <td>8.10000</td>\n",

" <td>8.100000</td>\n",

" <td>8.500000</td>\n",

" <td>8.200000</td>\n",

" <td>8.700000</td>\n",

" <td>8.800000</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"<p>8 rows × 53 columns</p>\n",

"</div>"

],

"text/plain": [

" title\_year budget Gross actor\_1\_facebook\_likes \\\n",

"count 100.000000 1.000000e+02 1.000000e+02 100.000000 \n",

"mean 2012.820000 7.838400e+07 1.468679e+08 13407.270000 \n",

"std 1.919491 7.445295e+07 1.454004e+08 10649.037862 \n",

"min 2010.000000 3.000000e+06 2.238380e+05 39.000000 \n",

"25% 2011.000000 1.575000e+07 4.199752e+07 1000.000000 \n",

"50% 2013.000000 4.225000e+07 1.070266e+08 13000.000000 \n",

"75% 2014.000000 1.500000e+08 2.107548e+08 20000.000000 \n",

"max 2016.000000 2.600000e+08 9.366622e+08 35000.000000 \n",

"\n",

" actor\_2\_facebook\_likes actor\_3\_facebook\_likes IMDb\_rating \\\n",

"count 99.000000 98.000000 100.000000 \n",

"mean 7377.303030 3002.153061 7.883000 \n",

"std 13471.568216 6940.301133 0.247433 \n",

"min 12.000000 0.000000 7.500000 \n",

"25% 580.000000 319.750000 7.700000 \n",

"50% 1000.000000 626.500000 7.800000 \n",

"75% 11000.000000 1000.000000 8.100000 \n",

"max 96000.000000 46000.000000 8.800000 \n",

"\n",

" MetaCritic Runtime CVotes10 ... Votes1829F Votes3044 \\\n",

"count 95.000000 100.000000 100.000000 ... 100.000000 100.000000 \n",

"mean 78.252632 126.420000 73212.160000 ... 7.982000 7.732000 \n",

"std 9.122066 19.050799 82669.594746 ... 0.321417 0.251814 \n",

"min 62.000000 91.000000 6420.000000 ... 7.300000 7.300000 \n",

"25% 72.000000 114.750000 30587.000000 ... 7.700000 7.600000 \n",

"50% 78.000000 124.000000 54900.500000 ... 8.000000 7.700000 \n",

"75% 83.500000 136.250000 80639.000000 ... 8.200000 7.900000 \n",

"max 100.000000 180.000000 584839.000000 ... 8.800000 8.700000 \n",

"\n",

" Votes3044M Votes3044F Votes45A Votes45AM Votes45AF Votes1000 \\\n",

"count 100.000000 100.000000 100.00000 100.000000 100.000000 100.000000 \n",

"mean 7.723000 7.780000 7.65100 7.624000 7.770000 7.274000 \n",

"std 0.260479 0.282128 0.21485 0.213258 0.301344 0.361987 \n",

"min 7.200000 7.200000 7.10000 7.100000 7.000000 6.400000 \n",

"25% 7.500000 7.600000 7.50000 7.475000 7.500000 7.100000 \n",

"50% 7.700000 7.800000 7.65000 7.600000 7.800000 7.300000 \n",

"75% 7.900000 8.000000 7.80000 7.800000 7.925000 7.500000 \n",

"max 8.700000 8.500000 8.10000 8.100000 8.500000 8.200000 \n",

"\n",

" VotesUS VotesnUS \n",

"count 100.000000 100.000000 \n",

"mean 7.958000 7.793000 \n",

"std 0.232327 0.264099 \n",

"min 7.500000 7.300000 \n",

"25% 7.800000 7.600000 \n",

"50% 7.950000 7.750000 \n",

"75% 8.100000 7.925000 \n",

"max 8.700000 8.800000 \n",

"\n",

"[8 rows x 53 columns]"

]

},

"execution\_count": 6,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# Check the summary for the numeric columns \n",

"\n",

"movies.describe()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Task 2: Data Analysis\n",

"\n",

"Now that we have loaded the dataset and inspected it, we see that most of the data is in place. As of now, no data cleaning is required, so let's start with some data manipulation, analysis, and visualisation to get various insights about the data. "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.1: Reduce those Digits!\n",

"\n",

"These numbers in the `budget` and `gross` are too big, compromising its readability. Let's convert the unit of the `budget` and `gross` columns from `$` to `million $` first."

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"metadata": {},

"outputs": [],

"source": [

"# Divide the 'gross' and 'budget' columns by 1000000 to convert '$' to 'million $'\n",

"movies.gross=round((movies.Gross/1000000),2) #Converting Gross to millions\n",

"movies.budget=round((movies.budget/1000000),2) #Converting Budget to millions \n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.2: Let's Talk Profit!\n",

"\n",

" 1. Create a new column called `profit` which contains the difference of the two columns: `gross` and `budget`.\n",

" 2. Sort the dataframe using the `profit` column as reference.\n",

" 3. Extract the top ten profiting movies in descending order and store them in a new dataframe - `top10`.\n",

" 4. Plot a scatter or a joint plot between the columns `budget` and `profit` and write a few words on what you observed.\n",

" 5. Extract the movies with a negative profit and store them in a new dataframe - `neg\_profit`"

]

},

{

"cell\_type": "code",

"execution\_count": 8,

"metadata": {},

"outputs": [],

"source": [

"# Create the new column named 'profit' by subtracting the 'budget' column from the 'gross' column\n",

"movies['profit']=movies.Gross-movies.budget # taking a difference of Gross and budget and storing the result in profit\n"

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {},

"outputs": [],

"source": [

"# Sort the dataframe with the 'profit' column as reference using the 'sort\_values' function. Make sure to set the argument\n",

"#'ascending' to 'False'\n",

"movies.sort\_values('profit',ascending=False, ignore\_index=True, inplace=True) #sorting using profit and resetting the index\n"

]

},

{

"cell\_type": "code",

"execution\_count": 10,

"metadata": {},

"outputs": [],

"source": [

"# Get the top 10 profitable movies by using position based indexing. Specify the rows till 10 (0-9)\n",

"\n",

"top10=movies.iloc[:10]"

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 720x576 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"#Plot profit vs budget\n",

"plt.figure(figsize=(10,8))\n",

"plt.title(\"budget v/s profit\", fontdict={'fontsize':15, 'fontweight':5, 'color':'Red'})\n",

"plt.scatter(movies.profit,movies.budget, alpha=0.7)\n",

"plt.xlabel('Profit',fontdict={'fontsize':12})\n",

"plt.ylabel('budget',fontdict={'fontsize':12})\n",

"plt.show()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"The dataset contains the 100 best performing movies from the year 2010 to 2016. However scatter plot tells a different story. You can notice that there are some movies with negative profit. Although good movies do incur losses, but there appear to be quite a few movie with losses. What can be the reason behind this? Lets have a closer look at this by finding the movies with negative profit."

]

},

{

"cell\_type": "code",

"execution\_count": 12,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"Series([], Name: Title, dtype: object)"

]

},

"execution\_count": 12,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"#Find the movies with negative profit\n",

"neg\_profit=movies[movies.profit<0]\n",

"neg\_profit.Title\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Checkpoint 1:`\*\* Can you spot the movie `Tangled` in the dataset? You may be aware of the movie 'Tangled'. Although its one of the highest grossing movies of all time, it has negative profit as per this result. If you cross check the gross values of this movie (link: https://www.imdb.com/title/tt0398286/), you can see that the gross in the dataset accounts only for the domestic gross and not the worldwide gross. This is true for may other movies also in the list."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.3: The General Audience and the Critics\n",

"\n",

"You might have noticed the column `MetaCritic` in this dataset. This is a very popular website where an average score is determined through the scores given by the top-rated critics. Second, you also have another column `IMDb\_rating` which tells you the IMDb rating of a movie. This rating is determined by taking the average of hundred-thousands of ratings from the general audience. \n",

"\n",

"As a part of this subtask, you are required to find out the highest rated movies which have been liked by critics and audiences alike.\n",

"1. Firstly you will notice that the `MetaCritic` score is on a scale of `100` whereas the `IMDb\_rating` is on a scale of 10. First convert the `MetaCritic` column to a scale of 10.\n",

"2. Now, to find out the movies which have been liked by both critics and audiences alike and also have a high rating overall, you need to -\n",

" - Create a new column `Avg\_rating` which will have the average of the `MetaCritic` and `Rating` columns\n",

" - Retain only the movies in which the absolute difference(using abs() function) between the `IMDb\_rating` and `Metacritic` columns is less than 0.5. Refer to this link to know how abs() funtion works - https://www.geeksforgeeks.org/abs-in-python/ .\n",

" - Sort these values in a descending order of `Avg\_rating` and retain only the movies with a rating equal to higher than `8` and store these movies in a new dataframe `UniversalAcclaim`.\n",

" "

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"metadata": {},

"outputs": [],

"source": [

"# Change the scale of MetaCritic\n",

"\n",

"movies['MetaCritic']=round(movies['MetaCritic']/10,2) #Dividing metacritic by 10 to convert in into a scale of 10"

]

},

{

"cell\_type": "code",

"execution\_count": 14,

"metadata": {},

"outputs": [],

"source": [

"# Find the average ratings\n",

"#calculating mean of metacritic and IMDB columns by setting axis to columns\n",

"movies['Avg\_rating']=movies[['MetaCritic','IMDb\_rating']].mean(axis=1)"

]

},

{

"cell\_type": "code",

"execution\_count": 15,

"metadata": {},

"outputs": [],

"source": [

"#Sort in descending order of average rating\n",

"movies.sort\_values('Avg\_rating', ascending=False, inplace=True)\n"

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"metadata": {},

"outputs": [],

"source": [

"# Find the movies with metacritic-rating < 0.5 and also with the average rating of >8\n",

"UniversalAcclaim=movies[((movies.MetaCritic-movies.IMDb\_rating.abs())<0.5)&(movies.Avg\_rating>=8)].sort\_values('Avg\_rating', ascending=False, ignore\_index=True)\n",

"\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Checkpoint 2:`\*\* Can you spot a `Star Wars` movie in your final dataset?yes"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.4: Find the Most Popular Trios - I\n",

"\n",

"You're a producer looking to make a blockbuster movie. There will primarily be three lead roles in your movie and you wish to cast the most popular actors for it. Now, since you don't want to take a risk, you will cast a trio which has already acted in together in a movie before. The metric that you've chosen to check the popularity is the Facebook likes of each of these actors.\n",

"\n",

"The dataframe has three columns to help you out for the same, viz. `actor\_1\_facebook\_likes`, `actor\_2\_facebook\_likes`, and `actor\_3\_facebook\_likes`. Your objective is to find the trios which has the most number of Facebook likes combined. That is, the sum of `actor\_1\_facebook\_likes`, `actor\_2\_facebook\_likes` and `actor\_3\_facebook\_likes` should be maximum.\n",

"Find out the top 5 popular trios, and output their names in a list.\n"

]

},

{

"cell\_type": "code",

"execution\_count": 17,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"5 most popular actor trios:\n"

]

},

{

"data": {

"text/plain": [

"[('Dev Patel', 'Nicole Kidman', 'Rooney Mara'),\n",

" ('Leonardo DiCaprio', 'Tom Hardy', 'Joseph Gordon-Levitt'),\n",

" ('Jennifer Lawrence', 'Peter Dinklage', 'Hugh Jackman'),\n",

" ('Casey Affleck', 'Michelle Williams ', 'Kyle Chandler'),\n",

" ('Tom Hardy', 'Christian Bale', 'Joseph Gordon-Levitt')]"

]

},

"execution\_count": 17,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# Write your code here\n",

"# Handling NaN values in actor\_1\_facebook\_likes, actor\_2\_facebook\_likes, actor\_3\_facebook\_likes with zeroes\n",

"movies.actor\_1\_facebook\_likes=np.where(movies.actor\_1\_facebook\_likes.isnull(),0,movies.actor\_1\_facebook\_likes)\n",

"movies.actor\_2\_facebook\_likes=np.where(movies.actor\_2\_facebook\_likes.isnull(),0,movies.actor\_2\_facebook\_likes)\n",

"movies.actor\_3\_facebook\_likes=np.where(movies.actor\_3\_facebook\_likes.isnull(),0,movies.actor\_3\_facebook\_likes)\n",

"\n",

"#Creating a trio\_actor\_likes field to calculate sum of likes of popular actors\n",

"movies['trio\_actor\_likes']=movies[['actor\_1\_facebook\_likes','actor\_2\_facebook\_likes','actor\_3\_facebook\_likes']].sum(axis=1)\n",

"\n",

"#Calculating 5 most popular actor trios\n",

"movies.groupby(['actor\_1\_name','actor\_2\_name','actor\_3\_name']).trio\_actor\_likes.sum().sort\_values(ascending=False).index[:5]\n",

"\n",

"#Appending and printing 5 popular actor trios to a list\n",

"popular\_trios=movies.groupby(['actor\_1\_name','actor\_2\_name','actor\_3\_name']).trio\_actor\_likes.sum().sort\_values(ascending=False).index[:5].to\_list()\n",

"\n",

"print('5 most popular actor trios:')\n",

"popular\_trios\n",

"\n",

"\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.5: Find the Most Popular Trios - II\n",

"\n",

"In the previous subtask you found the popular trio based on the total number of facebook likes. Let's add a small condition to it and make sure that all three actors are popular. The condition is \*\*none of the three actors' Facebook likes should be less than half of the other two\*\*. For example, the following is a valid combo:\n",

"- actor\_1\_facebook\_likes: 70000\n",

"- actor\_2\_facebook\_likes: 40000\n",

"- actor\_3\_facebook\_likes: 50000\n",

"\n",

"But the below one is not:\n",

"- actor\_1\_facebook\_likes: 70000\n",

"- actor\_2\_facebook\_likes: 40000\n",

"- actor\_3\_facebook\_likes: 30000\n",

"\n",

"since in this case, `actor\_3\_facebook\_likes` is 30000, which is less than half of `actor\_1\_facebook\_likes`.\n",

"\n",

"Having this condition ensures that you aren't getting any unpopular actor in your trio (since the total likes calculated in the previous question doesn't tell anything about the individual popularities of each actor in the trio.).\n",

"\n",

"You can do a manual inspection of the top 5 popular trios you have found in the previous subtask and check how many of those trios satisfy this condition. Also, which is the most popular trio after applying the condition above?"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*Write your answers below.\*\*\n",

"\n",

"- \*\*`No. of trios that satisfy the above condition:`\*\*\n",

"\n",

"- \*\*`Most popular trio after applying the condition:`\*\*"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Optional:`\*\* Even though you are finding this out by a natural inspection of the dataframe, can you also achieve this through some \*if-else\* statements to incorporate this. You can try this out on your own time after you are done with the assignment."

]

},

{

"cell\_type": "code",

"execution\_count": 18,

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"actor\_1\_name actor\_2\_name actor\_3\_name \n",

"Leonardo DiCaprio Tom Hardy Joseph Gordon-Levitt 79000.0\n",

"Jennifer Lawrence Peter Dinklage Hugh Jackman 76000.0\n",

"Tom Hardy Christian Bale Joseph Gordon-Levitt 73000.0\n",

"Chris Hemsworth Robert Downey Jr. Scarlett Johansson 66000.0\n",

"Robert Downey Jr. Scarlett Johansson Chris Evans 51000.0\n",

"Philip Seymour Hoffman Robin Wright Brad Pitt 51000.0\n",

"Hugh Jackman Eddie Redmayne Anne Hathaway 44000.0\n",

"Jeff Bridges James Franco Mackenzie Foy 29000.0\n",

"Matthew McConaughey Anne Hathaway Mackenzie Foy 28000.0\n",

"Jack McBrayer Sarah Silverman Joe Lo Truglio 2739.0\n",

"Madeline Carroll Rebecca De Mornay Aidan Quinn 2639.0\n",

"BÃ©rÃ©nice Bejo Ed Lauter Beth Grant 2521.0\n",

"Devin Ratray Bruce Dern Will Forte 2466.0\n",

"Clea DuVall Scoot McNairy Tate Donovan 2310.0\n",

"Patrick Fugit Sela Ward Emily Ratajkowski 2272.0\n",

"Damon Wayans Jr. Daniel Henney Abraham Benrubi 2037.0\n",

"Elizabeth McGovern Deborah Twiss Michael Rispoli 1426.0\n",

"Aldis Hodge Neil Brown Jr. R. Marcos Taylor 1289.0\n",

"Ellar Coltrane Lorelei Linklater Libby Villari 550.0\n",

"Elina Alminas Sonoya Mizuno Corey Johnson 417.0\n",

"Name: trio\_actor\_likes, dtype: float64"

]

},

"execution\_count": 18,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"# Your answer here (optional)\n",

"# Getting valid trios of actors with satisfying conditions\n",

"popular\_trios=movies[-((movies.actor\_1\_facebook\_likes<movies.actor\_3\_facebook\_likes/2) | ((movies.actor\_1\_facebook\_likes<movies.actor\_2\_facebook\_likes/2))\n",

" |(movies.actor\_2\_facebook\_likes<movies.actor\_3\_facebook\_likes/2) | ((movies.actor\_2\_facebook\_likes<movies.actor\_1\_facebook\_likes/2))\n",

" |(movies.actor\_3\_facebook\_likes<movies.actor\_1\_facebook\_likes/2) | ((movies.actor\_3\_facebook\_likes<movies.actor\_2\_facebook\_likes/2)))\n",

" ]\n",

"# Grouping and aggregating the likes of actors and sorting them\n",

"popular\_trios.groupby(['actor\_1\_name','actor\_2\_name','actor\_3\_name']).trio\_actor\_likes.sum().sort\_values(ascending=False)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.6: Runtime Analysis\n",

"\n",

"There is a column named `Runtime` in the dataframe which primarily shows the length of the movie. It might be intersting to see how this variable this distributed. Plot a `histogram` or `distplot` of seaborn to find the `Runtime` range most of the movies fall into."

]

},

{

"cell\_type": "code",

"execution\_count": 19,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Runtime histogram/density plot\n",

"sns.distplot(movies.Runtime)\n",

"plt.title('movie Runtime', fontsize=15)\n",

"plt.xlabel('Runtime in minuts', fontsize=14)\n",

"plt.ylabel('density', fontsize=14)\n",

"plt.show()\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Checkpoint 3:`\*\* Most of the movies appear to be sharply 2 hour-long."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 2.7: R-Rated Movies\n",

"\n",

"Although R rated movies are restricted movies for the under 18 age group, still there are vote counts from that age group. Among all the R rated movies that have been voted by the under-18 age group, find the top 10 movies that have the highest number of votes i.e.`CVotesU18` from the `movies` dataframe. Store these in a dataframe named `PopularR`."

]

},

{

"cell\_type": "code",

"execution\_count": 20,

"metadata": {},

"outputs": [],

"source": [

"# Write your code here\n",

"popularR=movies[(movies.content\_rating=='R') & (movies.CVotesU18>0)][['Title','CVotesU18']].sort\_values('CVotesU18', ascending=False).head(10)"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Checkpoint 4:`\*\* Are these kids watching `Deadpool` a lot?"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

" "

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"## Task 3 : Demographic analysis\n",

"\n",

"If you take a look at the last columns in the dataframe, most of these are related to demographics of the voters (in the last subtask, i.e., 2.8, you made use one of these columns - CVotesU18). We also have three genre columns indicating the genres of a particular movie. We will extensively use these columns for the third and the final stage of our assignment wherein we will analyse the voters across all demographics and also see how these vary across various genres. So without further ado, let's get started with `demographic analysis`."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 3.1 Combine the Dataframe by Genres\n",

"\n",

"There are 3 columns in the dataframe - `genre\_1`, `genre\_2`, and `genre\_3`. As a part of this subtask, you need to aggregate a few values over these 3 columns. \n",

"1. First create a new dataframe `df\_by\_genre` that contains `genre\_1`, `genre\_2`, and `genre\_3` and all the columns related to \*\*CVotes/Votes\*\* from the `movies` data frame. There are 47 columns to be extracted in total.\n",

"2. Now, Add a column called `cnt` to the dataframe `df\_by\_genre` and initialize it to one. You will realise the use of this column by the end of this subtask.\n",

"3. First group the dataframe `df\_by\_genre` by `genre\_1` and find the sum of all the numeric columns such as `cnt`, columns related to CVotes and Votes columns and store it in a dataframe `df\_by\_g1`.\n",

"4. Perform the same operation for `genre\_2` and `genre\_3` and store it dataframes `df\_by\_g2` and `df\_by\_g3` respectively. \n",

"5. Now that you have 3 dataframes performed by grouping over `genre\_1`, `genre\_2`, and `genre\_3` separately, it's time to combine them. For this, add the three dataframes and store it in a new dataframe `df\_add`, so that the corresponding values of Votes/CVotes get added for each genre.There is a function called `add()` in pandas which lets you do this. You can refer to this link to see how this function works. https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.add.html\n",

"6. The column `cnt` on aggregation has basically kept the track of the number of occurences of each genre.Subset the genres that have atleast 10 movies into a new dataframe `genre\_top10` based on the `cnt` column value.\n",

"7. Now, take the mean of all the numeric columns by dividing them with the column value `cnt` and store it back to the same dataframe. We will be using this dataframe for further analysis in this task unless it is explicitly mentioned to use the dataframe `movies`.\n",

"8. Since the number of votes can't be a fraction, type cast all the CVotes related columns to integers. Also, round off all the Votes related columns upto two digits after the decimal point.\n"

]

},

{

"cell\_type": "code",

"execution\_count": 21,

"metadata": {},

"outputs": [],

"source": [

"# Create the dataframe df\_by\_genre\n",

"df\_by\_genre=movies[['genre\_1','genre\_2','genre\_3','CVotes10','CVotes09','CVotes08','CVotes07','CVotes06',\n",

" 'CVotes05','CVotes04','CVotes03','CVotes02','CVotes01','CVotesMale','CVotesFemale','CVotesU18',\n",

" 'CVotesU18M','CVotesU18F','CVotes1829','CVotes1829M','CVotes1829F','CVotes3044','CVotes3044M',\n",

" 'CVotes3044F','CVotes45A','CVotes45AM','CVotes45AF','CVotes1000','CVotesUS','CVotesnUS','VotesM',\n",

" 'VotesF','VotesU18','VotesU18M','VotesU18F','Votes1829','Votes1829M','Votes1829F','Votes3044',\n",

" 'Votes3044M','Votes3044F','Votes45A','Votes45AM','Votes45AF','Votes1000','VotesUS','VotesnUS']]"

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"metadata": {},

"outputs": [],

"source": [

"# Create a column cnt and initialize it to 1\n",

"\n",

"df\_by\_genre['cnt']=1"

]

},

{

"cell\_type": "code",

"execution\_count": 23,

"metadata": {},

"outputs": [],

"source": [

"# Group the movies by individual genres\n",

"df\_by\_g1=df\_by\_genre.groupby(['genre\_1']).sum()\n",

"df\_by\_g2=df\_by\_genre.groupby(['genre\_2']).sum()\n",

"df\_by\_g3=df\_by\_genre.groupby(['genre\_3']).sum()"

]

},

{

"cell\_type": "code",

"execution\_count": 24,

"metadata": {},

"outputs": [],

"source": [

"# Add the grouped data frames and store it in a new data frame\n",

"\n",

"df\_add=df\_by\_g1.add(df\_by\_g2, axis=0, fill\_value=0).add(df\_by\_g3, axis=1, fill\_value=0) # adding g2 to g1 and g3 to g1+g2"

]

},

{

"cell\_type": "code",

"execution\_count": 25,

"metadata": {},

"outputs": [],

"source": [

"# Extract genres with atleast 10 occurences\n",

"\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": 26,

"metadata": {},

"outputs": [],

"source": [

"# Take the mean for every column by dividing with cnt \n",

"\n",

"#Storing top 10 genres in a new dataframe - genre\_top10\n",

"genre\_top10=df\_add[df\_add.cnt>10]\n",

"\n",

"#Storing counts of movies - cnt in temp variable - top10\_cnt\n",

"top10\_cnt=genre\_top10.cnt\n",

"\n",

"#Taking the mean of all columns by with respect to cnt\n",

"genre\_top=genre\_top10[:].apply(lambda x: x.div(genre\_top10.cnt))\n",

"\n",

"#Updating the cnt field with values in temp variable - top10cnt\n",

"genre\_top10['cnt']=top10\_cnt"

]

},

{

"cell\_type": "code",

"execution\_count": 27,

"metadata": {},

"outputs": [],

"source": [

"# Rounding off the columns of Votes to two decimals\n",

"\n",

"genre\_top10=genre\_top10.loc[:].round(2)"

]

},

{

"cell\_type": "code",

"execution\_count": 28,

"metadata": {},

"outputs": [],

"source": [

"# Converting CVotes to int type\n",

"\n",

"#Storing CVotes columns in a list\n",

"Cvotes\_col=[]\n",

"for c in genre\_top10:\n",

" if c.startswith('Cvotes'):\n",

" Cvotes\_col.append(c)\n",

"#Convert CVote column types to a int\n",

"genre\_top10[Cvotes\_col]=genre\_top10[Cvotes\_col].astype('int')"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"If you take a look at the final dataframe that you have gotten, you will see that you now have the complete information about all the demographic (Votes- and CVotes-related) columns across the top 10 genres. We can use this dataset to extract exciting insights about the voters!"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 3.2: Genre Counts!\n",

"\n",

"Now let's derive some insights from this data frame. Make a bar chart plotting different genres vs cnt using seaborn."

]

},

{

"cell\_type": "code",

"execution\_count": 29,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 648x504 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Countplot for genres\n",

"plt.figure(figsize=[9,7])\n",

"sns.barplot(data=genre\_top10, x=genre\_top10.index, y='cnt')\n",

"plt.xlabel('Genre', fontsize=14)\n",

"plt.ylabel('Movies', fontsize=14)\n",

"plt.tight\_layout()\n",

"plt.show()\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Checkpoint 5:`\*\* Is the bar for `Drama` the tallest?yes\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 3.3: Gender and Genre\n",

"\n",

"If you have closely looked at the Votes- and CVotes-related columns, you might have noticed the suffixes `F` and `M` indicating Female and Male. Since we have the vote counts for both males and females, across various age groups, let's now see how the popularity of genres vary between the two genders in the dataframe. \n",

"\n",

"1. Make the first heatmap to see how the average number of votes of males is varying across the genres. Use seaborn heatmap for this analysis. The X-axis should contain the four age-groups for males, i.e., `CVotesU18M`,`CVotes1829M`, `CVotes3044M`, and `CVotes45AM`. The Y-axis will have the genres and the annotation in the heatmap tell the average number of votes for that age-male group. \n",

"\n",

"2. Make the second heatmap to see how the average number of votes of females is varying across the genres. Use seaborn heatmap for this analysis. The X-axis should contain the four age-groups for females, i.e., `CVotesU18F`,`CVotes1829F`, `CVotes3044F`, and `CVotes45AF`. The Y-axis will have the genres and the annotation in the heatmap tell the average number of votes for that age-female group. \n",

"\n",

"3. Make sure that you plot these heatmaps side by side using `subplots` so that you can easily compare the two genders and derive insights.\n",

"\n",

"4. Write your any three inferences from this plot. You can make use of the previous bar plot also here for better insights.\n",

"Refer to this link- https://seaborn.pydata.org/generated/seaborn.heatmap.html. You might have to plot something similar to the fifth chart in this page (You have to plot two such heatmaps side by side).\n",

"\n",

"5. Repeat subtasks 1 to 4, but now instead of taking the CVotes-related columns, you need to do the same process for the Votes-related columns. These heatmaps will show you how the two genders have rated movies across various genres.\n",

"\n",

"You might need the below link for formatting your heatmap.\n",

"https://stackoverflow.com/questions/56942670/matplotlib-seaborn-first-and-last-row-cut-in-half-of-heatmap-plot\n",

"\n",

"- Note : Use `genre\_top10` dataframe for this subtask"

]

},

{

"cell\_type": "code",

"execution\_count": 30,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 864x360 with 2 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# 1st set of heat maps for CVotes-related columns\n",

"\n",

"plt.figure(figsize=(12,5))\n",

"\n",

"#Creating a subplot for Males Vote counts\n",

"plt.subplot(1,2,1)\n",

"plt.title('Male vote counts Heatmap', fontsize=15)\n",

"\n",

"#Creating a male heatmap using a pivot\n",

"sns.heatmap(genre\_top10.pivot\_table(index=genre\_top10.index, aggfunc='mean')[['CVotesU18M','CVotes1829M','CVotes3044M','CVotes45AM']],annot=True, cmap='RdYlGn', fmt='1g')\n",

"\n",

"#Creating a subplot for Female Vote counts\n",

"plt.subplot(1,2,1)\n",

"plt.title('Female vote counts Heatmap', fontsize=15)\n",

"#Creating a female heatmap using a pivot\n",

"sns.heatmap(genre\_top10.pivot\_table(index=genre\_top10.index, aggfunc='mean')[['CVotesU18F','CVotes1829F','CVotes3044F','CVotes45AF']],annot=True, cmap='RdYlGn', fmt='1g')\n",

"\n",

"#Aligning the heatmaps\n",

"plt.tight\_layout()\n",

"plt.show()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Inferences:`\*\* A few inferences that can be seen from the heatmap above is that males have voted more than females, and Sci-Fi appears to be most popular among the 18-29 age group irrespective of their gender. What more can you infer from the two heatmaps that you have plotted? Write your three inferences/observations below:\n",

"- Inference 1:\n",

"- Inference 2:\n",

"- Inference 3:"

]

},

{

"cell\_type": "code",

"execution\_count": 31,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 864x360 with 4 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# 2nd set of heat maps for Votes-related columns\n",

"\n",

"plt.figure(figsize=(12,5))\n",

"\n",

"#Creating a subplot for Males Ratings\n",

"plt.subplot(1,2,1)\n",

"plt.title('Male Ratings Heatmap', fontsize=15)\n",

"#Creating a male heatmap using a pivot\n",

"sns.heatmap(genre\_top10.pivot\_table(index=genre\_top10.index, aggfunc='mean')[['VotesU18M','Votes1829M','Votes3044M','Votes45AM']],annot=True, cmap='RdYlGn', fmt='1g')\n",

"plt.xticks(rotation=90)\n",

"\n",

"#Creating a subplot for Female Ratings\n",

"plt.subplot(1,2,2)\n",

"plt.title('Female Ratings Heatmap', fontsize=15)\n",

"#Creating a female heatmap using a pivot\n",

"sns.heatmap(genre\_top10.pivot\_table(index=genre\_top10.index, aggfunc='mean')[['VotesU18F','Votes1829F','Votes3044F','Votes45AF']],annot=True, cmap='RdYlGn', fmt='1g')\n",

"plt.xticks(rotation=90)\n",

"\n",

"#Aligning the heatmaps\n",

"plt.tight\_layout()\n",

"plt.show()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Inferences:`\*\* Sci-Fi appears to be the highest rated genre in the age group of U18 for both males and females. Also, females in this age group have rated it a bit higher than the males in the same age group. What more can you infer from the two heatmaps that you have plotted? Write your three inferences/observations below:\n",

"- Inference 1:under-18 age group rates the movies better than other age groups even when the propotion of their votes is lower.\n",

"- Inference 2:the ratings decrease steadily as the age increases for both genders. however, the average ratings of females are higher than males.\n",

"- Inference 3:animation genre has the most consistent rating among females of all age groups"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 3.4: US vs non-US Cross Analysis\n",

"\n",

"The dataset contains both the US and non-US movies. Let's analyse how both the US and the non-US voters have responded to the US and the non-US movies.\n",

"\n",

"1. Create a column `IFUS` in the dataframe `movies`. The column `IFUS` should contain the value \"USA\" if the `Country` of the movie is \"USA\". For all other countries other than the USA, `IFUS` should contain the value `non-USA`.\n",

"\n",

"\n",

"2. Now make a boxplot that shows how the number of votes from the US people i.e. `CVotesUS` is varying for the US and non-US movies. Make use of the column `IFUS` to make this plot. Similarly, make another subplot that shows how non US voters have voted for the US and non-US movies by plotting `CVotesnUS` for both the US and non-US movies. Write any of your two inferences/observations from these plots.\n",

"\n",

"\n",

"3. Again do a similar analysis but with the ratings. Make a boxplot that shows how the ratings from the US people i.e. `VotesUS` is varying for the US and non-US movies. Similarly, make another subplot that shows how `VotesnUS` is varying for the US and non-US movies. Write any of your two inferences/observations from these plots.\n",

"\n",

"Note : Use `movies` dataframe for this subtask. Make use of this documention to format your boxplot - https://seaborn.pydata.org/generated/seaborn.boxplot.html"

]

},

{

"cell\_type": "code",

"execution\_count": 32,

"metadata": {},

"outputs": [],

"source": [

"# Creating IFUS column\n",

"\n",

"movies[\"IFUS\"]=np.where(movies['Country']=='USA','USA','non-USA')"

]

},

{

"cell\_type": "code",

"execution\_count": 33,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Box plot - 1: CVotesUS(y) vs IFUS(x)\n",

"\n",

"plt.title('country v/s US voters analysis', fontsize=15)\n",

"sns.boxplot(data=movies, x=movies.IFUS, y=movies.CVotesUS)\n",

"plt.xlabel('country', fontsize='14')\n",

"plt.ylabel('count if US votes', fontsize='14')\n",

"plt.show()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Inferences:`\*\* Write your two inferences/observations below:\n",

"- Inference 1:The number of votes on non - USA movies are very uniformly distributed compared to USA movies.\n",

"- Inference 2:USA voters voted dominantly for USA movies. They voted less for non - USA movies"

]

},

{

"cell\_type": "code",

"execution\_count": 34,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Box plot - 2: VotesUS(y) vs IFUS(x)\n",

"plt.title('country v/s non-US voters analysis', fontsize=15)\n",

"sns.boxplot(data=movies, x=movies.IFUS, y=movies.CVotesnUS)\n",

"plt.xlabel('Country', fontsize='14')\n",

"plt.ylabel('Count of non-US votes', fontsize='14')\n",

"plt.show()\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Inferences:`\*\* Write your two inferences/observations below:\n",

"- Inference 1:Non - USA voters voted more in numbers for USA movies. This could be because of high number of USA movies in our dataset.\n",

"- Inference 2:The number of votes from non - USA people are higher than votes from USA people. This can be inferred from the scale in plots 1 and 2."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"- ### Subtask 3.5: Top 1000 Voters Vs Genres\n",

"\n",

"You might have also observed the column `CVotes1000`. This column represents the top 1000 voters on IMDb and gives the count for the number of these voters who have voted for a particular movie. Let's see how these top 1000 voters have voted across the genres. \n",

"\n",

"1. Sort the dataframe genre\_top10 based on the value of `CVotes1000`in a descending order.\n",

"\n",

"2. Make a seaborn barplot for `genre` vs `CVotes1000`.\n",

"\n",

"3. Write your inferences. You can also try to relate it with the heatmaps you did in the previous subtasks.\n",

"\n",

"\n"

]

},

{

"cell\_type": "code",

"execution\_count": 35,

"metadata": {},

"outputs": [],

"source": [

"# Sorting by CVotes1000\n",

"genre\_top10.sort\_values('CVotes1000', ascending=False, inplace=True)\n"

]

},

{

"cell\_type": "code",

"execution\_count": 36,

"metadata": {},

"outputs": [

{

"data": {

"image/png": "\n",

"text/plain": [

"<Figure size 576x432 with 1 Axes>"

]

},

"metadata": {

"needs\_background": "light"

},

"output\_type": "display\_data"

}

],

"source": [

"# Bar plot\n",

"plt.figure(figsize=(8,6))\n",

"plt.title('Genre v/s CVotes1000', fontsize=14)\n",

"sns.barplot(data=genre\_top10, x=genre\_top10.index, y='CVotes1000')\n",

"plt.xlabel('Genre', fontsize=14)\n",

"plt.ylabel('CVotes1000', fontsize=14)\n",

"plt.tight\_layout()\n",

"plt.show()"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Inferences:`\*\* Write your inferences/observations here.\n",

"\*Inference 1: Sci-Fi is the most popular among the top 1000 voters. This is in-line with count of votes received from other voters.\n",

"\*Inference 2: Romance is the least voted genre. Since males voted the least for this genre, we can try to see the gender split among the top 1000 voters.\n",

"\*\n",

"Inference 3: The popularity of Comedy, Biography and Drama genres are almost the same among top 1000 voters.\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\*\*`Checkpoint 6:`\*\* The genre `Romance` seems to be most unpopular among the top 1000 voters."

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"\n",

"\n"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"With the above subtask, your assignment is over. In your free time, do explore the dataset further on your own and see what kind of other insights you can get across various other columns.\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3 (ipykernel)",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.9.12"

}

},

"nbformat": 4,

"nbformat\_minor": 2

}