In [1]:
```
import pandas as pd
import numpy as np
from calendar import monthrange
```

In [2]:
```
twitch_df = pd.read_csv('Twitch_game_data.csv', encoding='cp1252')
```

In [3]:
```
twitch_df.head()
```

Out[3]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_chann
0	1	League of Legends	1	2016	94377226	1362044	530270	29
1	2	Counter-Strike: Global Offensive	1	2016	47832863	830105	372654	2
2	3	Dota 2	1	2016	45185893	433397	315083	1
3	4	Hearthstone	1	2016	39936159	235903	131357	5
4	5	Call of Duty: Black Ops III	1	2016	16153057	1151578	71639	36

In [4]:
```
twitch_df['Month'].describe()
```

Out[4]:
```
count    17400.000000
mean         6.344828
std          3.493370
min          1.000000
25%          3.000000
50%          6.000000
75%          9.000000
max         12.000000
Name: Month, dtype: float64
```

In [5]:
```
twitch_df_global = pd.read_csv('Twitch_global_data.csv', encoding='cp1252')
```

In [6]:
```
twitch_df_global.head()
```

Out[6]:

	year	Month	Hours_watched	Avg_viewers	Peak_viewers	Streams	Avg_channels	Games_st
0	2016	1	480241904	646355	1275257	7701675	20076	
1	2016	2	441859897	635769	1308032	7038520	20427	
2	2016	3	490669308	660389	1591551	7390957	20271	
3	2016	4	377975447	525696	1775120	6869719	16791	
4	2016	5	449836631	605432	1438962	7535519	19394	

In [7]:
```
twitch_df.describe()
```

Out[7]:

	Rank	Month	Year	Hours_watched	Hours_streamed	Peak_viewers
count	17400.000000	17400.000000	17400.000000	1.740000e+04	1.740000e+04	1.740000e+04
mean	100.500000	6.344828	2019.137931	5.406489e+06	1.740947e+05	6.129193e+04
std	57.735964	3.493370	2.096430	1.847645e+07	5.546933e+05	1.479578e+05
min	1.000000	1.000000	2016.000000	8.981100e+04	1.900000e+01	4.410000e+02
25%	50.750000	3.000000	2017.000000	4.329688e+05	1.414950e+04	9.270000e+03
50%	100.500000	6.000000	2019.000000	9.594830e+05	3.669700e+04	2.228400e+04
75%	150.250000	9.000000	2021.000000	2.690568e+06	9.990650e+04	5.158575e+04
max	200.000000	12.000000	2023.000000	3.445520e+08	1.024570e+07	3.366021e+06

In [8]:
```
'''RNC and DNC are single events, I need to look out for other single events a
```

Out[8]:
```
'RNC and DNC are single events, I need to look out for other single events an
d remove them from the dataset.'
```

In [9]:
```
twitch_df[twitch_df['Peak_channels'] == 1]
```

Out[9]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
1329	130	RNC 2016	7	2016	264303	28	29021	

In [10]:
```
twitch_df[twitch_df['Game'] == "DNC 2016"]
```

Out[10]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
1287	88	DNC 2016	7	2016	362702	355	47549	24

In [11]:
```
twitch_df.drop([1329, 1287], inplace = True)
twitch_df.reset_index(inplace = True, drop = True)
```

In [12]:
```
#game_list is long, but you can review after downloading code and typing it. C
```

In [13]:
```
game_list = list(set(list(twitch_df['Game'])))
```

In [14]:
```
'''There were a number of things to review/fix from the the game list on twitc
-- all E3 and twitchcon events are single events and would skew data
-- special characters not appearing correctly are to be fixed
-- review near duplicate games, Final Fantasy XIV and Legend of Zelda Games ha

    As an example there are three versions of Legend of Zelda: Ocarina of time
    and tagged as different games so these will not be combined in any fashion

    Final Fantasy XIV: Online has several different iterations of names. Most
    be consistent. Final Fantasy XIV: Online was chosen. There was one instanc
    with Final Fantasy XIV: Online. If it was just a matter of this one overla
    that Heavensward is simply an expansion of Final Fantasy Online, Final Fan
    Final Fantasy XIV: Online and the one month of overlapping data will be co
    erroneously categorized as having dropped out of the top 200

'''
```

Out[14]:
```
'There were a number of things to review/fix from the the game list on twitch
\n-- all E3 and twitchcon events are single events and would skew data\n-- sp
ecial characters not appearing correctly are to be fixed\n-- review near dupl
icate games, Final Fantasy XIV and Legend of Zelda Games have some titles tha
t are almost identical. \n    \n    As an example there are three versions of
Legend of Zelda: Ocarina of time. After some research these are still being s
treamed\n    and tagged as different games so these will not be combined in a
ny fashion. With a similar rationale I did not combine others\n    \n    Fina
l Fantasy XIV: Online has several different iterations of names. Most are jus
t a function of capitalization and need to\n    be consistent. Final Fantasy
XIV: Online was chosen. There was one instance of Final Fantasy XIV: Heavensw
ard overlapping \n    with Final Fantasy XIV: Online. If it was just a matter
of this one overlapping month I would have left it as is. But given\n    that
Heavensward is simply an expansion of Final Fantasy Online, Final Fantasy XI
V: Heavensward will be changed to \n    Final Fantasy XIV: Online and the one
month of overlapping data will be combined. This is because otherwise this wo
uld be \n    erroneously categorized as having dropped out of the top 200\n
\n'
```

In [15]:
```
twitch_df[twitch_df['Game'].str.contains('TwitchCon') == True]
```

Out[15]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_char
1761	164	TwitchCon 2016	9	2016	119450	166	44758	
1848	51	TwitchCon 2016	10	2016	829794	725	125427	
4259	62	TwitchCon 2017	10	2017	878473	1602	86285	

In [16]:
```
twitch_df.drop([1761,1848,4259], inplace = True)
```

In [17]:
```
ff_df = twitch_df[twitch_df['Game'].str.contains('XIV') == True]
ff_df.head()
```

Out[17]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_ch
111	112	Final Fantasy XIV: Heavensward	1	2016	305520	33688	13084	
284	85	Final Fantasy XIV: Heavensward	2	2016	358212	37285	3246	
464	65	Final Fantasy XIV: Heavensward	3	2016	598914	43341	17436	
688	89	Final Fantasy XIV: Heavensward	4	2016	319118	39940	5466	
885	86	Final Fantasy XIV: Heavensward	5	2016	305008	36029	10178	

In [18]:
```
ff_list = list(set(list(ff_df['Game'])))
```

In [19]:
```
ff_list
```

Out[19]:
```
['FINAL FANTASY XIV ONLINE',
 'Final Fantasy XIV: Heavensward',
 'FINAL FANTASY XIV Online',
 'Romance of the Three Kingdoms XIV',
 'The King of Fighters XIV',
 'Final Fantasy XIV Online']
```

In [20]:
```
ff_list.remove('Romance of the Three Kingdoms XIV')
ff_list.remove('The King of Fighters XIV')
```

In [21]:
```
ff_list
```

Out[21]:
```
['FINAL FANTASY XIV ONLINE',
 'Final Fantasy XIV: Heavensward',
 'FINAL FANTASY XIV Online',
 'Final Fantasy XIV Online']
```

In [22]:
```
twitch_df.reset_index(drop = True, inplace = True)
```

In [23]:
```
twitch_df = twitch_df.replace(ff_list, 'Final Fantasy XIV: Online')
```

In [24]:
```
e3_df = twitch_df[twitch_df['Game'].str.contains('E3') == True]
e3_df.index
```

Out[24]:
```
Int64Index([1005, 3402, 3497, 5801, 5848, 5896], dtype='int64')
```

In [25]:
```
e3_df.describe()
```

Out[25]:

	Rank	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_cl
count	6.000000	6.0	6.000000	6.000000e+00	6.000000	6.000000e+00	6
mean	46.333333	6.0	2017.333333	8.987702e+06	18801.333333	8.606043e+05	1698
std	46.855807	0.0	0.816497	9.571019e+06	24962.272186	5.482893e+05	1843
min	6.000000	6.0	2016.000000	3.814990e+05	184.000000	3.882860e+05	177
25%	7.000000	6.0	2017.000000	8.738742e+05	1327.750000	4.375315e+05	408
50%	30.500000	6.0	2017.500000	7.549571e+06	9975.000000	7.290035e+05	1263
75%	90.000000	6.0	2018.000000	1.414050e+07	24437.750000	1.051961e+06	1993
max	102.000000	6.0	2018.000000	2.345944e+07	64970.000000	1.817345e+06	5125

In [26]:
```
twitch_df.drop(e3_df.index, inplace = True)
```

In [27]:
```
twitch_df.drop(twitch_df[twitch_df['Game'] == 'Twitch Presents'].index, inplac
twitch_df.reset_index(inplace = True, drop = True)
```

In [28]:
```
'''One thing that surprises me is the amount of classic games in this dataset.
that games that are popular are not necessarily new so having a recent release
popularity.
'''
```

Out[28]:
```
'One thing that surprises me is the amount of classic games in this dataset.
This will change the nature of the analysis in \nthat games that are popular
are not necessarily new so having a recent release date will likely negativel
y impact future \npopularity. \n'
```

In [29]:
```
twitch_df[twitch_df['Game'] =='Phoenix Wright: Ace Attorney <U+2212> Spirit of
```

Out[29]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_char
1791	196	Phoenix Wright: Ace Attorney <U+2212> Spirit o...	9	2016	96620	842	2287	

In [30]:
```
twitch_df = twitch_df.replace('Phoenix Wright: Ace Attorney <U+2212> Spirit of
```

In [31]:
```
twitch_df[twitch_df['Game'] =='Phoenix Wright: Ace Attorney - Spirit of Justic
```

Out[31]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_chann
1791	196	Phoenix Wright: Ace Attorney - Spirit of Justice	9	2016	96620	842	2287	

In [32]:
```
poke_df = twitch_df[twitch_df['Game'].str.contains('Pok') == True]
poke_df.head(10)
```

Out[32]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Pea
21	22	Poker	1	2016	3151063	36445	18034	
50	51	Pok<U+00E9>mon Omega Ruby/Alpha Sapphire	1	2016	992216	20237	11278	
78	79	Pok<U+00E9>mon Red/Blue	1	2016	456370	3276	185106	
79	80	Pok<U+00E9>mon Yellow	1	2016	449878	855	210369	
226	27	Poker	2	2016	2465052	36936	14039	
252	53	Pok<U+00E9>mon Omega Ruby/Alpha Sapphire	2	2016	1014081	17894	16439	
321	122	Pok<U+00E9>mon Red/Blue	2	2016	192855	4838	20162	
377	178	Pok<U+00E9>mon FireRed/LeafGreen	2	2016	117710	3260	18360	
422	23	Poker	3	2016	3534525	41966	25720	
444	45	Pok<U+00E9>mon Omega Ruby/Alpha Sapphire	3	2016	1246547	24192	6980	

In [33]:
```
poke_list = list(set(list(poke_df['Game'])))
```

In [34]:
```
poke_list.remove('Poker')
poke_list.remove('Prominence Poker')
```

In [35]:
```
twitch_df.reset_index(drop = True, inplace = True)
```

In [36]:
```
def clean (string):
    lst = []
    for letter in string:
        lst.append(letter)
    if '<' in lst:
        index = lst.index('<')
        del lst[index:index + 8]
        lst.insert(index, 'e')
        lst = ''.join(lst)
        return(lst)
    if 'Ã' in lst:
        index = lst.index('Ã')
        del lst[index:index + 2]
        lst.insert(index, 'e')
        lst = ''.join(lst)
        return(lst)
    else:
        lst = ''.join(lst)
        return(lst)
```

In [37]:
```
x = len(twitch_df)
for i in range(x):
    twitch_df['Game'][i] = str(twitch_df['Game'][i])
    twitch_df['Game'][i] = clean(twitch_df['Game'][i])
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\1143704737.py:3: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df['Game'][i] = str(twitch_df['Game'][i])
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\1143704737.py:4: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df['Game'][i] = clean(twitch_df['Game'][i])
```

In [38]:
```
poke_list[0]
```

Out[38]:
```
'PokÃ©mon GO'
```

In [39]:
```
poke_df = twitch_df[twitch_df['Game'].str.contains('Pok') == True]
poke_df.head(10)
```

Out[39]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Pea
21	22	Poker	1	2016	3151063	36445	18034	
50	51	Pokemon Omega Ruby/Alpha Sapphire	1	2016	992216	20237	11278	
78	79	Pokemon Red/Blue	1	2016	456370	3276	185106	
79	80	Pokemon Yellow	1	2016	449878	855	210369	
226	27	Poker	2	2016	2465052	36936	14039	
252	53	Pokemon Omega Ruby/Alpha Sapphire	2	2016	1014081	17894	16439	
321	122	Pokemon Red/Blue	2	2016	192855	4838	20162	
377	178	Pokemon FireRed/LeafGreen	2	2016	117710	3260	18360	
422	23	Poker	3	2016	3534525	41966	25720	
444	45	Pokemon Omega Ruby/Alpha Sapphire	3	2016	1246547	24192	6980	

In [40]:
```
twitch_df.head()
```

Out[40]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_chann
0	1	League of Legends	1	2016	94377226	1362044	530270	29
1	2	Counter-Strike: Global Offensive	1	2016	47832863	830105	372654	2
2	3	Dota 2	1	2016	45185893	433397	315083	1
3	4	Hearthstone	1	2016	39936159	235903	131357	5
4	5	Call of Duty: Black Ops III	1	2016	16153057	1151578	71639	36

In [41]:
```
#As a result of fixing the pokemon games God of War Ragnarok was changed, but
```

In [42]:
```
gow_df = twitch_df[twitch_df['Game'].str.contains('Ragnarek') == True]
```

In [43]:
```
gow_df
```

Out[43]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_cha
16392	11	God of War Ragnarek	11	2022	40440752	1617975	480655	
16637	56	God of War Ragnarek	12	2022	3798888	477077	34187	
16869	88	God of War Ragnarek	1	2023	1727833	257951	26831	
17175	194	God of War Ragnarek	2	2023	552004	122533	7011	

In [44]:
```
twitch_df = twitch_df.replace('God of War Ragnarek', 'God of War Ragnarok')
```

In [45]:
```
#As a result of fixing the pokemon games Okami was changed, but incorrectly, t
```

In [46]:
```
okami_df = twitch_df[twitch_df['Game'].str.contains('kami') == True]
okami_df
```

Out[46]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_char
4773	185	ekami	12	2017	181539	19273	29429	
13887	106	Demon Slayer - Kimetsu no Yaiba- The Hinokami C...	10	2021	1641397	93397	57275	

In [47]:
```
twitch_df = twitch_df.replace('ekami', 'Okami')
```

In [48]:
```
twitch_df.reset_index(drop = True, inplace = True)
```

In [49]:
```
twitch_df = twitch_df.sort_values(by = ['Game', 'Year', 'Month'])
twitch_df.head(10)
twitch_df.reset_index(drop = True, inplace = True)
```

In [50]:
```
x = len(twitch_df)
dup_l = []
for i in range(x-1):
    if twitch_df['Game'][i] == twitch_df['Game'][i+1] and twitch_df['Month'][i
        dup_l.append(i)
```

In [51]:
```
dup_l
```

Out[51]:
```
[518, 872, 3894, 4970, 11111, 11120, 11173, 11289, 11291, 13076]
```

In [52]:
```
examine = twitch_df.filter(items = dup_l, axis=0)
examine = examine.sort_values(by = ['Game', 'Year', 'Month'])
```

In [53]:
```
'''As we can see by the dataframe below there are a lot of duplicates, these w
One challenge in this is that Peak_viewers, Peak_channels, and Streamers are n
I will choose the max value of each.

For consistency I should compute the average viewers, channels, and viewer_rat
on the other columns in the dataframe for all columns.'''
```

Out[53]:
```
'As we can see by the dataframe below there are a lot of duplicates, these wi
ll also have to be combined in some fashion.\nOne challenge in this is that P
eak_viewers, Peak_channels, and Streamers are not computable. \nI will choose
the max value of each. \n\nFor consistency I should compute the average viewe
rs, channels, and viewer_ratio based \non the other columns in the dataframe
for all columns.'
```

In [54]:
```
examine
```

Out[54]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_cha
518	68	Among Us	1	2023	2760266	76925	137287	
872	15	Atlas	1	2019	12110031	187032	69614	
3894	75	Dungeons & Dragons	9	2018	790966	21961	38597	
4970	86	Final Fantasy XIV: Online	5	2017	533590	28073	35091	
11111	116	Resident Evil	10	2016	251183	3976	9871	
11120	13	Resident Evil 2	1	2019	16394917	449333	314244	
11173	11	Resident Evil 4	3	2023	27426481	1054133	327946	
11289	72	Retro	9	2021	2976840	131077	20554	
11291	83	Retro	10	2021	2484523	97914	10543	
13076	27	Stray	7	2022	11428809	508590	287683	

In [55]:
```
new_values= []
def add_col(col):
    new_values.clear()
    for i in dup_l:
        new_values.append(twitch_df[col][i] + twitch_df[col][i+1])
```

In [56]:
```
add_col('Hours_watched')
```

In [57]:
```
x = len(new_values)
def replace_cols(cols):
    for i in range(x):
        twitch_df.replace(twitch_df[cols][dup_l[i]], new_values[i], inplace =
```

In [58]:
```
replace_cols('Hours_watched')
```

In [59]:
```
twitch_df['Hours_watched'][518]
```

Out[59]: 3636990

In [60]:
```
add_col('Hours_streamed')
```

In [61]:
```
replace_cols('Hours_streamed')
```

In [62]:
```
def higher(col):
    new_values.clear()
    for i in dup_l:
        if twitch_df[col][i] > twitch_df[col][i+1]:
            new_values.append(twitch_df[col][i])
        else:
            new_values.append(twitch_df[col][i+1])
```

In [63]:
```
higher('Peak_viewers')
```

In [64]:
```
replace_cols('Peak_viewers')
```

In [65]:
```
higher('Peak_channels')
```

In [66]:
```
higher('Streamers')
```

In [67]:
```
'''
(Hours watched/hours in the month) is approximately the same as the current Av
(Hours streamed/hours in the month) is approximately the same as Avg_channels;

Avg_viewer_ratio is (Avg_viewers/Avg_channels); however when computing with th
by 0. To use these values I simply reversed the division to Avg_channels/Avg_v

I will be using these values instead given there are some duplicates not of my
Also, the data card was not forthcoming as to the provenance of these columns.
makes sense.

Rank will not be helpful at this point so I will remove'''
```

Out[67]:
```
'\n(Hours watched/hours in the month) is approximately the same as the curren
t Avg_viewers; \n(Hours streamed/hours in the month) is approximately the sam
e as Avg_channels;\n\nAvg_viewer_ratio is (Avg_viewers/Avg_channels); however
when computing with the values I found that I was sometimes dividing\nby 0. T
o use these values I simply reversed the division to Avg_channels/Avg_viewer
s.\n\nI will be using these values instead given there are some duplicates no
t of my own making that needed combined.\nAlso, the data card was not forthco
ming as to the provenance of these columns. Creating my own similar metrics\n
makes sense. \n\nRank will not be helpful at this point so I will remove'
```

In [68]:
```
no_leap = []
leap = []
for i in range(1,13):
    leap.append(monthrange(2004, i)[1] * 24)
    no_leap.append(monthrange(2003, i)[1] * 24)
```

In [69]:
```
no_leap
```

Out[69]:
```
[744, 672, 744, 720, 744, 720, 744, 744, 720, 744, 720, 744]
```

In [70]:
```
leap
```

Out[70]:
```
[744, 696, 744, 720, 744, 720, 744, 744, 720, 744, 720, 744]
```

In [71]:
```
x = len(twitch_df)
month_hours = []
for i in range(x):
    if twitch_df['Year'][i] %4:
        hours = leap[(twitch_df['Month'][i]-1)]
        month_hours.append(hours)
    else:
        hours = no_leap[(twitch_df['Month'][i]-1)]
        month_hours.append(hours)
```

In [72]:
```
twitch_df['Month_hours'] = month_hours
twitch_df.head()
```

Out[72]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channe
0	199	.hack//G.U. Last Recode	11	2017	145350	35258	1222	2
1	159	20 Minutes Till Dawn	6	2022	911356	12253	29743	
2	109	60 Parsecs!	9	2018	529688	1867	31960	
3	126	60 Seconds!	7	2016	268754	597	32505	
4	54	60 Seconds!	8	2016	772786	2065	56904	

In [73]:
```
twitch_df['Avg_viewers'] = round(twitch_df['Hours_watched']/twitch_df['Month_h
twitch_df.head()
```

Out[73]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channe
0	199	.hack//G.U. Last Recode	11	2017	145350	35258	1222	25
1	159	20 Minutes Till Dawn	6	2022	911356	12253	29743	8
2	109	60 Parsecs!	9	2018	529688	1867	31960	2
3	126	60 Seconds!	7	2016	268754	597	32505	1
4	54	60 Seconds!	8	2016	772786	2065	56904	3

In [74]:
```
twitch_df['Avg_channels'] = round(twitch_df['Hours_streamed']/twitch_df['Month
twitch_df.head()
```

Out[74]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channe
0	199	.hack//G.U. Last Recode	11	2017	145350	35258	1222	25
1	159	20 Minutes Till Dawn	6	2022	911356	12253	29743	8
2	109	60 Parsecs!	9	2018	529688	1867	31960	2
3	126	60 Seconds!	7	2016	268754	597	32505	1
4	54	60 Seconds!	8	2016	772786	2065	56904	3

In [75]:
```
twitch_df['Avg_channel_ratio'] = round(twitch_df['Avg_channels']/twitch_df['Av
twitch_df.head()
```

Out[75]:

	Rank	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channe
0	199	.hack//G.U. Last Recode	11	2017	145350	35258	1222	2
1	159	20 Minutes Till Dawn	6	2022	911356	12253	29743	
2	109	60 Parsecs!	9	2018	529688	1867	31960	
3	126	60 Seconds!	7	2016	268754	597	32505	
4	54	60 Seconds!	8	2016	772786	2065	56904	

In [76]:
```
twitch_df.drop(['Rank', 'Month_hours', 'Avg_viewer_ratio'], axis = 1, inplace
```

In [77]:
```
twitch_df.drop(twitch_df[twitch_df['Game'] == 'Twitch Presents'].index, inplac
twitch_df.reset_index(inplace = True, drop = True)
```

In [78]:
```
twitch_df.sort_values(by = ['Month', 'Year', 'Game'], inplace = True)
twitch_df.head(20)
```

Out[78]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
12	7 Days to Die	1	2016	269681	12131	4405	44
258	Agar.io	1	2016	255617	20705	4183	74
264	Age of Empires	1	2016	248884	232	107455	18
422	Alien: Isolation	1	2016	264294	11799	9590	42
477	American Truck Simulator	1	2016	314055	724	43089	48
694	Ark: Survival Evolved	1	2016	1951875	93060	19486	241
718	Arma 3	1	2016	2542838	86219	32132	275
911	Azure Striker GUNVOLT	1	2016	197178	217	135933	14
928	Banjo-Kazooie	1	2016	241250	2234	108131	28
967	BattleBlock Theater	1	2016	332256	2041	152739	19
1011	Battlefield 4	1	2016	672524	96185	6349	271
1139	Black Desert Online	1	2016	320675	15798	1888	44
1227	Blade & Soul	1	2016	5270251	184279	67256	1099
1268	Blast Corps	1	2016	178810	42	96318	18
1287	Bloodborne	1	2016	1300353	86384	199060	257
1403	Borderlands 2	1	2016	175135	21849	3376	71
1465	Brain Age: Train Your Brain in Minutes a Day!	1	2016	173011	19	207721	14
1661	Call of Duty: Black Ops II	1	2016	309850	29294	4491	100
1679	Call of Duty: Black Ops III	1	2016	16153057	1151578	71639	3620
2031	Cities: Skylines	1	2016	172577	8602	7371	30

In [79]:
```
df = list(twitch_df.Year.astype(str) + '/' + twitch_df.Month.astype(str) + '/0
twitch_df['Date'] = df
twitch_df['Date']= pd.to_datetime(twitch_df['Date'])
```

In [80]:
```
twitch_df_X = twitch_df.drop_duplicates(subset=['Game'])
twitch_df_X.head(10)
```

Out[80]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	S
12	7 Days to Die	1	2016	269681	12131	4405	44	
258	Agar.io	1	2016	255617	20705	4183	74	
264	Age of Empires	1	2016	248884	232	107455	18	
422	Alien: Isolation	1	2016	264294	11799	9590	42	
477	American Truck Simulator	1	2016	314055	724	43089	48	
694	Ark: Survival Evolved	1	2016	1951875	93060	19486	241	
718	Arma 3	1	2016	2542838	86219	32132	275	
911	Azure Striker GUNVOLT	1	2016	197178	217	135933	14	
928	Banjo-Kazooie	1	2016	241250	2234	108131	28	
967	BattleBlock Theater	1	2016	332256	2041	152739	19	

In [81]:
```
twitch_df_X.describe()
```

Out[81]:

	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
count	2025.000000	2025.000000	2.025000e+03	2.025000e+03	2.025000e+03	2025.00000
mean	4.402469	2018.737778	1.722624e+06	5.265007e+04	5.925928e+04	342.25777
std	3.545371	2.247062	5.395172e+06	2.188540e+05	9.022342e+04	1071.92884
min	1.000000	2016.000000	9.246000e+04	1.900000e+01	7.470000e+02	2.00000
25%	1.000000	2017.000000	2.719870e+05	2.857000e+03	1.644000e+04	25.00000
50%	3.000000	2018.000000	6.132720e+05	1.046700e+04	3.332000e+04	76.00000
75%	7.000000	2021.000000	1.235417e+06	3.050200e+04	6.725600e+04	233.00000
max	12.000000	2023.000000	9.437723e+07	4.948200e+06	1.832845e+06	29306.00000

In [82]:
```
zero_df = twitch_df[twitch_df['Avg_channels'] == 0]
zero_df.head(11)
```

Out[82]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
264	Age of Empires	1	2016	248884	232	107455	18
911	Azure Striker GUNVOLT	1	2016	197178	217	135933	14
1268	Blast Corps	1	2016	178810	42	96318	18
1465	Brain Age: Train Your Brain in Minutes a Day!	1	2016	173011	19	207721	14
6404	Half Life 2: Survivor	1	2016	400106	195	164561	17
7098	Iji	1	2016	179556	30	91449	16
7470	Kirby 64: The Crystal Shards	1	2016	315368	209	158893	16
7478	Kirby: Squeak Squad	1	2016	299628	60	156564	16
8690	Mega Man 10	1	2016	197038	141	158783	29
8862	Mike Tyson's Punch-Out!!	1	2016	197881	307	168929	30
9502	Ninja Gaiden III: The Ancient Ship of Doom	1	2016	183550	122	133273	16

In [83]:
```
one_df = twitch_df[twitch_df['Avg_channels'] == 1]
one_df.head(11)
```

Out[83]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
477	American Truck Simulator	1	2016	314055	724	43089	48
2354	Crash Bandicoot	1	2016	399886	501	159957	29
2865	Darksiders II	1	2016	242099	752	120528	11
3650	Donkey Kong Country	1	2016	243490	826	160369	17
3652	Donkey Kong Country 2: Diddy's Kong Quest	1	2016	316101	993	181740	28
5059	Fire Emblem: Path of Radiance	1	2016	235555	516	122456	10
6411	Halo 4	1	2016	302218	569	135671	13
7983	M.U.G.E.N	1	2016	386581	944	976	5
8211	Mafia LIVE!	1	2016	183378	772	9532	8
8531	Mario Kart 64	1	2016	173981	605	186466	21
8788	Metroid Fusion	1	2016	234149	587	147009	22

In [84]:
```
two_df = twitch_df[twitch_df['Avg_channels'] == 2]
two_df.head(11)
```

Out[84]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	S
4878	Final Fantasy IV	1	2016	535452	1158	147048	21	
5974	Grand Theft Auto III	1	2016	216252	1118	142392	22	
6141	Grid 2	1	2016	176713	1173	11348	9	
8694	Mega Man 2	1	2016	208756	1841	150406	121	
8697	Mega Man X	1	2016	180986	1415	181667	22	
8781	Metro: Last Light	1	2016	186279	1309	30280	13	
8790	Metroid Prime	1	2016	248704	1136	150677	25	
8958	Mirror's Edge	1	2016	209507	1528	186567	35	
9158	Move or Die	1	2016	202884	1275	32211	12	
10053	Paper Mario	1	2016	412453	1272	115905	16	
13447	Super Mario Bros. 3	1	2016	426084	1161	206252	17	

In [85]:
```
plus_one = 1
X = twitch_df_X.Date + pd.DateOffset(months=plus_one)
```

In [86]:
```
twitch_df_X['one_month_future'] =  twitch_df_X.Date + pd.DateOffset(months=plu
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\2229078347.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df_X['one_month_future'] =  twitch_df_X.Date + pd.DateOffset(months=
plus_one)
```

In [87]:
```
twitch_df_X.head()
```

Out[87]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	St
12	7 Days to Die	1	2016	269681	12131	4405	44	
258	Agar.io	1	2016	255617	20705	4183	74	
264	Age of Empires	1	2016	248884	232	107455	18	
422	Alien: Isolation	1	2016	264294	11799	9590	42	
477	American Truck Simulator	1	2016	314055	724	43089	48	

In [88]:
```
plus_three = 3
X3 = twitch_df_X.Date + pd.DateOffset(months=plus_three)
```

In [89]:
```
twitch_df_X['three_month_future'] =  twitch_df_X.Date + pd.DateOffset(months=p
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\4009328254.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df_X['three_month_future'] =  twitch_df_X.Date + pd.DateOffset(month
s=plus_three)
```

In [90]:
```
twitch_df_X.head()
```

Out[90]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	St
12	7 Days to Die	1	2016	269681	12131	4405	44	
258	Agar.io	1	2016	255617	20705	4183	74	
264	Age of Empires	1	2016	248884	232	107455	18	
422	Alien: Isolation	1	2016	264294	11799	9590	42	
477	American Truck Simulator	1	2016	314055	724	43089	48	

In [91]:
```
plus_six = 6
X6 = twitch_df_X.Date + pd.DateOffset(months=plus_six)
```

In [92]:
```
twitch_df_X['six_month_future'] =  twitch_df_X.Date + pd.DateOffset(months=plu
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\3018229530.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df_X['six_month_future'] =  twitch_df_X.Date + pd.DateOffset(months=
plus_six)
```

In [93]:
```
twitch_df_X.head()
```

Out[93]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	St
12	7 Days to Die	1	2016	269681	12131	4405	44	
258	Agar.io	1	2016	255617	20705	4183	74	
264	Age of Empires	1	2016	248884	232	107455	18	
422	Alien: Isolation	1	2016	264294	11799	9590	42	
477	American Truck Simulator	1	2016	314055	724	43089	48	

In [94]:
```
twitch_df_X.reset_index(drop = True, inplace = True)
```

In [95]:
```
#twitch_df_X one_mnth_hrs = twitch_df['hours_watched'] where twitch_df['Date']
# if it doesn't exist make it 0
# current strategy append the index and value as a dict, when there is a missi
#and make the list the column hours_watched_omf
def f_hrs_watch (og_col_name, end_col_name):
    month_i = []
    month = []
    x = len(twitch_df_X)
    y = len(twitch_df)
    for i in range(x):
        for j in range(y):
            if twitch_df_X[og_col_name][i] == twitch_df['Date'][j] and twitch_
                month.append(twitch_df['Hours_watched'][j])
                month_i.append(i)
    zeros = []
    x = len(twitch_df_X)
    for i in range(x):
        if i not in month_i:
            zeros.append((i,0))
    merged_mth = list(zip(month_i, month))
    merged_z_mth = (merged_mth + zeros)
    merged_z_mth.sort()
    fin_mth = []
    for i in range(x):
        fin_mth.append(merged_z_mth[i][1])
    twitch_df_X[end_col_name] = fin_mth
```

In [96]:
```
f_hrs_watch('one_month_future', 'Hours_watched_1mth')
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\2513141890.py:26: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df_X[end_col_name] = fin_mth
```

In [97]:
```
twitch_df_X.describe()
```

Out[97]:

	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
count	2025.000000	2025.000000	2.025000e+03	2.025000e+03	2.025000e+03	2025.000000
mean	4.402469	2018.737778	1.722624e+06	5.265007e+04	5.925928e+04	342.257778
std	3.545371	2.247062	5.395172e+06	2.188540e+05	9.022342e+04	1071.928845
min	1.000000	2016.000000	9.246000e+04	1.900000e+01	7.470000e+02	2.000000
25%	1.000000	2017.000000	2.719870e+05	2.857000e+03	1.644000e+04	25.000000
50%	3.000000	2018.000000	6.132720e+05	1.046700e+04	3.332000e+04	76.000000
75%	7.000000	2021.000000	1.235417e+06	3.050200e+04	6.725600e+04	233.000000
max	12.000000	2023.000000	9.437723e+07	4.948200e+06	1.832845e+06	29306.000000

In [98]:
```
twitch_df_X[twitch_df_X['Hours_watched_1mth'] != 0].describe()
```

Out[98]:

	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
count	809.000000	809.000000	8.090000e+02	8.090000e+02	8.090000e+02	809.000000
mean	3.138443	2018.784920	3.173144e+06	1.082643e+05	6.265248e+04	613.195303
std	2.998039	2.272461	8.149991e+06	3.359434e+05	9.823830e+04	1566.070656
min	1.000000	2016.000000	9.975600e+04	1.420000e+02	7.470000e+02	5.000000
25%	1.000000	2017.000000	4.677800e+05	9.610000e+03	1.388000e+04	58.000000
50%	1.000000	2019.000000	9.968500e+05	2.447200e+04	3.505000e+04	157.000000
75%	5.000000	2021.000000	2.443756e+06	7.501500e+04	6.746900e+04	545.000000
max	11.000000	2023.000000	9.437723e+07	4.948200e+06	1.249796e+06	29306.000000

In [99]:
```
f_hrs_watch('three_month_future', 'Hours_watched_3mth')
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\2513141890.py:26: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df_X[end_col_name] = fin_mth
```

In [100]:
```
f_hrs_watch('six_month_future', 'Hours_watched_6mth')
```

```
C:\Users\danie\AppData\Local\Temp\ipykernel_3888\2513141890.py:26: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  twitch_df_X[end_col_name] = fin_mth
```

In [101]:
```
twitch_df_X[twitch_df_X['Hours_watched_3mth'] != 0].describe()
```

Out[101]:

	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
count	408.000000	408.000000	4.080000e+02	4.080000e+02	408.000000	408.000000
mean	1.610294	2018.200980	4.026296e+06	1.419502e+05	58421.044118	572.259804
std	1.603924	2.132883	1.022881e+07	3.902082e+05	98120.193932	1298.057629
min	1.000000	2016.000000	9.975600e+04	7.230000e+02	747.000000	5.000000
25%	1.000000	2016.000000	3.932162e+05	1.096575e+04	8568.250000	53.750000
50%	1.000000	2018.000000	1.018976e+06	3.184700e+04	25098.000000	136.000000
75%	1.000000	2020.000000	3.134386e+06	1.077270e+05	60421.750000	449.500000
max	9.000000	2022.000000	9.437723e+07	4.948200e+06	837094.000000	13547.000000

In [102]:
```
twitch_df_X[twitch_df_X['Hours_watched_6mth'] != 0].describe()
```

Out[102]:

	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
count	331.000000	331.000000	3.310000e+02	3.310000e+02	331.000000	331.000000
mean	1.247734	2018.193353	4.408722e+06	1.589651e+05	60451.353474	578.939577
std	0.758159	2.134582	1.117714e+07	4.267634e+05	104835.481296	1334.597625
min	1.000000	2016.000000	1.041330e+05	5.390000e+02	976.000000	5.000000
25%	1.000000	2016.000000	4.295595e+05	1.326300e+04	8420.500000	52.000000
50%	1.000000	2018.000000	1.025337e+06	3.705600e+04	23330.000000	137.000000
75%	1.000000	2020.000000	3.166135e+06	1.188490e+05	60488.500000	505.500000
max	6.000000	2022.000000	9.437723e+07	4.948200e+06	837094.000000	13547.000000

In [105]:
```
twitch_df.to_pickle('twitch_df_og.pkl')
```

In [106]:
```
twitch_df_X.to_pickle('twitch_df_wrng.pkl')
```

In [ ]:
```

```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sweetviz as sv
import warnings
```

In [2]:
```
warnings.filterwarnings('ignore')
```

In [3]:
```
twitch_df_X = pd.read_pickle('twitch_df_wrng.pkl')
```

In [4]:
```
twitch_df_X.head()
```

Out[4]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Strea
0	7 Days to Die	1	2016	269681	12131	4405	44	
1	Agar.io	1	2016	255617	20705	4183	74	
2	Age of Empires	1	2016	248884	232	107455	18	
3	Alien: Isolation	1	2016	264294	11799	9590	42	
4	American Truck Simulator	1	2016	314055	724	43089	48	

In [5]:
```
twitch_df = pd.read_pickle('twitch_df_og.pkl')
```

In [6]:
```
twitch_df.head()
```

Out[6]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	St
12	7 Days to Die	1	2016	269681	12131	4405	44	
258	Agar.io	1	2016	255617	20705	4183	74	
264	Age of Empires	1	2016	248884	232	107455	18	
422	Alien: Isolation	1	2016	264294	11799	9590	42	
477	American Truck Simulator	1	2016	314055	724	43089	48	

In [7]:
```
twitch_df_X[(twitch_df_X['Hours_watched_1mth'] == 0) & (twitch_df_X['Hours_wat
```

Out[7]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channe
94	Metroid Prime	1	2016	248704	1136	150677	
151	Super Mario Bros. 3	1	2016	426084	1161	206252	
155	Super Mario World	1	2016	270879	2888	150670	
156	Super Metroid	1	2016	576156	3905	191257	
166	The Elder Scrolls IV: Oblivion	1	2016	195502	1723	9447	
203	ArcheAge	1	2017	246054	14811	3868	

In [8]:
```
twitch_df_X[(twitch_df_X['Hours_watched_1mth'] == 0) & (twitch_df_X['Hours_wat
```

Out[8]:

	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
count	35.000000	35.000000	3.500000e+01	35.000000	35.000000	35.000000
mean	2.171429	2018.114286	4.944313e+05	13915.885714	55058.314286	107.285714
std	1.962677	1.966954	4.452449e+05	14098.685022	61231.169945	157.518920
min	1.000000	2016.000000	1.041330e+05	846.000000	2614.000000	7.000000
25%	1.000000	2017.000000	2.190645e+05	3521.000000	7725.500000	24.500000
50%	1.000000	2017.000000	3.448180e+05	9461.000000	27536.000000	57.000000
75%	2.000000	2019.500000	5.212395e+05	18359.500000	74112.500000	87.500000
max	8.000000	2022.000000	2.224158e+06	58877.000000	191257.000000	691.000000

In [9]:
```
report = sv.analyze(twitch_df_X)
report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

```
                                                |          | [  0%]   00:00 ->
(? left)
```

In [10]:
```
report = sv.analyze(twitch_df)
report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

```
                                                |          | [  0%]   00:00 ->
(? left)
```

In [11]:
```
'''This library is actually kind of amazing. I found this when I was strugglin
A big thing is that the first time a game breaks into the top 200 is typically
01-01-2016, no surprise there as that is when the dataset starts by definition
However, second was 01-01-2017, third was 01-01-2018 etc.... Almost half of th
and 01-01-2016 only accounts for 10%. Do these games remain popular in past Ja
```

Out[11]:
```
'This library is actually kind of amazing. I found this when I was struggling
to get ydata_profiling to work properly. \nA big thing is that the first time
a game breaks into the top 200 is typically January. The most common date was
\n01-01-2016, no surprise there as that is when the dataset starts by definit
ion that has the max value of 200. \nHowever, second was 01-01-2017, third wa
s 01-01-2018 etc.... Almost half of the games break into the top 200 in Janua
ry \nand 01-01-2016 only accounts for 10%. Do these games remain popular in p
ast January or are these more likely to be fads?'
```

In [12]:
```
jan_df= twitch_df_X[twitch_df_X['Month'] == 1]
jan_df.head(10)
```

Out[12]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Str
0	7 Days to Die	1	2016	269681	12131	4405	44	
1	Agar.io	1	2016	255617	20705	4183	74	
2	Age of Empires	1	2016	248884	232	107455	18	
3	Alien: Isolation	1	2016	264294	11799	9590	42	
4	American Truck Simulator	1	2016	314055	724	43089	48	
5	Ark: Survival Evolved	1	2016	1951875	93060	19486	241	
6	Arma 3	1	2016	2542838	86219	32132	275	
7	Azure Striker GUNVOLT	1	2016	197178	217	135933	14	
8	Banjo-Kazooie	1	2016	241250	2234	108131	28	
9	BattleBlock Theater	1	2016	332256	2041	152739	19	

In [13]:
```
report = sv.analyze(jan_df)
report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

```
                                                |          | [  0%]   00:00 ->
(? left)
```

In [14]:
```
x = len(twitch_df_X)
jan_list = []
for i in range(x):
    if twitch_df_X['Month'][i] == 1:
        jan_list.append(1)
    else:
        jan_list.append(0)
```

In [15]:
```
twitch_df_X['Jan_Debut_Month'] = jan_list
```

In [16]:
```
twitch_df_X.head()
```

Out[16]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Strea
0	7 Days to Die	1	2016	269681	12131	4405	44	
1	Agar.io	1	2016	255617	20705	4183	74	
2	Age of Empires	1	2016	248884	232	107455	18	
3	Alien: Isolation	1	2016	264294	11799	9590	42	
4	American Truck Simulator	1	2016	314055	724	43089	48	

In [17]:
```
report = sv.analyze(twitch_df_X)
report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

```
                                                |          | [  0%]   00:00 ->
(? left)
```

In [18]:
```
'''Based on the EDA in this dataset it seems that if a game debuted in January
Let me remove 2016 and see if the correlation still holds
Also of note there is a .95 correlation bewteen if a game is popular in the fi
thus a prediction on hours watched 6 months from debut is not required'''
```

Out[18]:
```
'Based on the EDA in this dataset it seems that if a game debuted in January
it will be popular one month later. \nLet me remove 2016 and see if the corre
lation still holds\nAlso of note there is a .95 correlation bewteen if a game
is popular in the first 3 months (hours watched) and 6 months,\nthus a predic
tion on hours watched 6 months from debut is not required'
```

In [19]:
```
no_2016_df = twitch_df_X[twitch_df_X['Year']!= 2016]
no_2016_df.head()
```

Out[19]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
200	ARK: Survival Evolved	1	2017	2167646	192501	18756	483
201	ASTRONEER	1	2017	761112	21225	29721	72
202	Age of Empires II	1	2017	310965	5299	4129	16
203	ArcheAge	1	2017	246054	14811	3868	43
204	Assassin's Creed II	1	2017	341584	3049	33045	22

In [20]:
```
report = sv.analyze(no_2016_df)
report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

```
                                                |          | [  0%]   00:00 ->
(? left)
```

In [21]:
```
'''the correlation still holds, this may be a useful feature, and a surprising
```

Out[21]:
```
'the correlation still holds, this may be a useful feature, and a surprising
one at that'
```

In [22]:
```
x = len(twitch_df_X)
top200_list = []
for i in range(x):
    if twitch_df_X['Hours_watched_1mth'][i] != 0:
        top200_list.append(1)
    else:
        top200_list.append(0)
```

In [23]:
```
twitch_df_X['Next_mth_200'] = top200_list
```

In [24]:
```
twitch_df_X.head()
```

Out[24]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Strea
0	7 Days to Die	1	2016	269681	12131	4405	44	
1	Agar.io	1	2016	255617	20705	4183	74	
2	Age of Empires	1	2016	248884	232	107455	18	
3	Alien: Isolation	1	2016	264294	11799	9590	42	
4	American Truck Simulator	1	2016	314055	724	43089	48	

In [25]:
```
report = sv.analyze(twitch_df_X)
report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

```
                                             |          | [  0%]   00:00 ->
(? left)
```

In [27]:
```
twitch_df_X.to_pickle('twitch_df_wrng.pkl')
```

In [ ]:
```

```

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ParameterGrid
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import mean_absolute_percentage_error
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import seaborn as sns
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
import sweetviz as sv
import warnings
```

In [2]:
```
warnings.filterwarnings('ignore')
```

In [3]:
```
twitch_df_X = pd.read_pickle('twitch_df_wrng.pkl')
```

In [4]:
```
twitch_df_X.head(2)
```

Out[4]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Stream
0	7 Days to Die	1	2016	269681	12131	4405	44	1:
1	Agar.io	1	2016	255617	20705	4183	74	4

In [5]:
```
twitch_df_X = twitch_df_X[twitch_df_X['Date'] != '2023-03-01']
```

In [6]:
```
twitch_df_X[(twitch_df_X['Year'] == 2023)].tail(2)
```

Out[6]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	S
895	Undisputed	2	2023	1066525	24598	69575	173	
896	Wild Hearts	2	2023	4637613	132860	76574	845	

In [7]:
```
twitch_df_X[twitch_df_X['Hours_watched_1mth'] == 0].count()
```

Out[7]:
```
Game                 1193
Month                1193
Year                 1193
Hours_watched        1193
Hours_streamed       1193
Peak_viewers         1193
Peak_channels        1193
Streamers            1193
Avg_viewers          1193
Avg_channels         1193
Avg_channel_ratio    1193
Date                 1193
one_month_future     1193
three_month_future   1193
six_month_future     1193
Hours_watched_1mth   1193
Hours_watched_3mth   1193
Hours_watched_6mth   1193
Jan_Debut_Month      1193
Next_mth_200         1193
dtype: int64
```

In [8]:
```
'''Baseline model would predict that ~60% of games will not be in the top 200
809 games were still in the top 200 after 1 month. Here I am setting up baseli
to read when the analysis of this baseline prediction and model prediction are
```

Out[8]:
```
'Baseline model would predict that ~60% of games will not be in the top 200 a
fter 1. \n809 games were still in the top 200 after 1 month. Here I am settin
g up baseline for comparison later. Notebook is easier\nto read when the anal
ysis of this baseline prediction and model prediction are by each other'
```

In [9]:
```
baseline_df = twitch_df_X.sort_values(by = 'Hours_watched', ascending = False)
```

In [10]:
```
baseline_df.reset_index(drop = True, inplace = True)
```

In [11]:
```
base_pred = [1]*809 + [0]*1193
```

In [12]:
```
baseline_df['prediction'] = base_pred
```

In [13]:
```
baseline_df.head(2)
```

Out[13]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Strea
0	League of Legends	1	2016	94377226	1362044	530270	2903	12
1	Escape from Tarkov	1	2022	93334264	1343113	690725	3186	7

2 rows × 21 columns

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(twitch_df_X.drop(columns=[
                                                    twitch_df_X.Next_mth_200,
                                                    random_state=1701)
```

In [15]:
```
c_report = sv.compare([X_train, "Training Data"], [X_test, "Test Data"])
```

```
                                                |          | [  0%]   00:00 ->
(? left)
```

In [16]:
```
'Test and training data do not appear to be different by any obvious criteria'
```

Out[16]:
```
'Test and training data do not appear to be different by any obvious criteri
a'
```

In [17]:
```
c_report.show_notebook(  w=None,
                h=None,
                scale=None,
                layout='widescreen',
                filepath=None)
```

In [18]:
```
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.c
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns = X_test.columr
X_train_scaled.head()
```

Out[18]:

	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Streamers	Avg_viewers	Avg
0	-0.104183	-0.114826	-0.127525	0.018430	-0.076539	-0.100087	
1	-0.264915	-0.207033	-0.153653	-0.278338	-0.234524	-0.264257	
2	-0.236449	-0.058641	-0.593625	-0.190841	-0.110451	-0.237468	
3	-0.271640	-0.212662	-0.519034	-0.282849	-0.264694	-0.268653	
4	-0.245541	-0.214101	-0.095324	-0.261200	-0.260094	-0.246398	

In [19]:
```
lr=LogisticRegression()
lr.fit(X_train_scaled, y_train)
```

Out[19]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

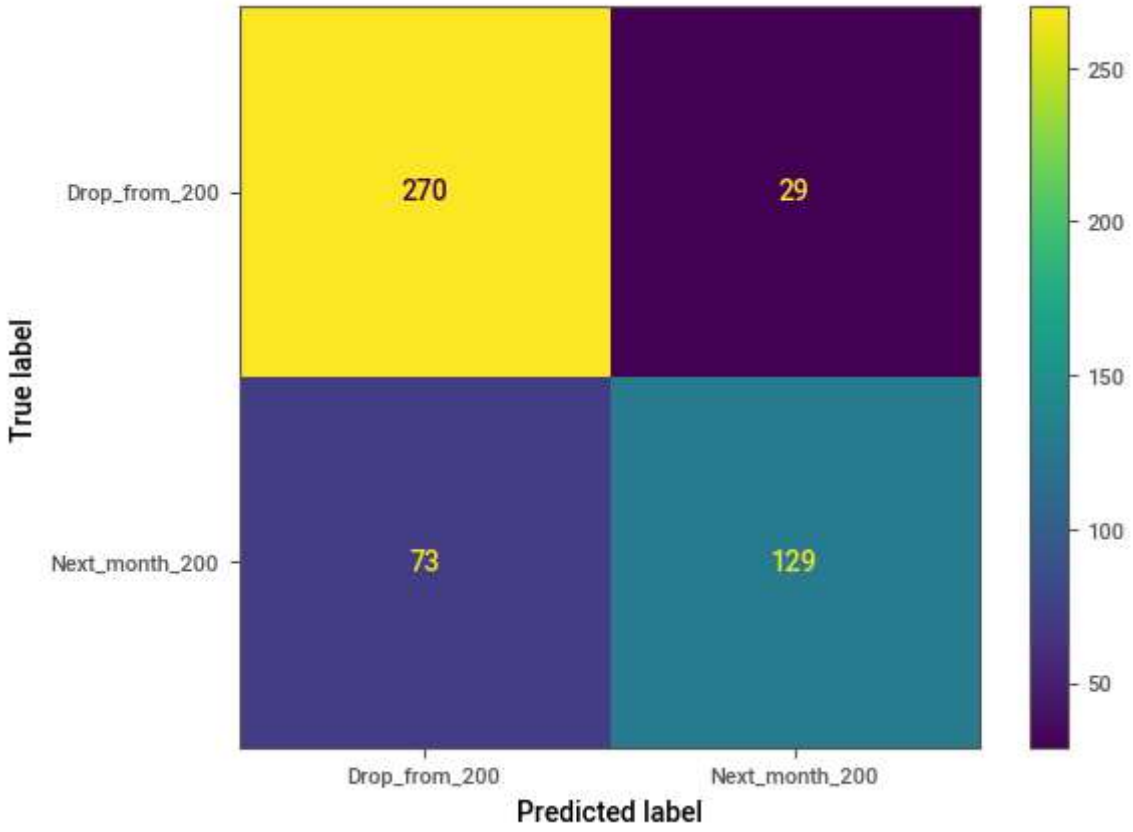
In [20]:
```
print("score on test: " + str(lr.score(X_test_scaled, y_test)))
print("score on train: "+ str(lr.score(X_train_scaled, y_train)))
```

```
score on test: 0.7964071856287425
score on train: 0.7321785476349101
```

In [21]:
```
y_pred_test_lr = lr.predict(X_test_scaled)
y_pred_train_lr = lr.predict(X_train_scaled)
```

In [22]:
```
cm = confusion_matrix(y_test, y_pred_test_lr, labels = lr.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = ['Drop_from_200', 'Next_m
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
ax.set_yticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
display_cm.plot(ax = ax)
```

Out[22]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a1eeaa01
90>
```



In [23]:
```
'''Prediction based only on hours watched is not as good as logistic regressio
```

Out[23]:
```
'Prediction based only on hours watched is not as good as logistic regression
or Random Forest Classifier'
```

In [24]:
```
print(classification_report(baseline_df['Next_mth_200'], baseline_df['predicti
```

```
              precision    recall  f1-score   support

           0       0.73      0.73      0.73      1193
           1       0.61      0.61      0.61       809

    accuracy                           0.68      2002
   macro avg       0.67      0.67      0.67      2002
weighted avg       0.68      0.68      0.68      2002
```

In [25]:
```
print(classification_report(y_test, y_pred_test_lr))
```

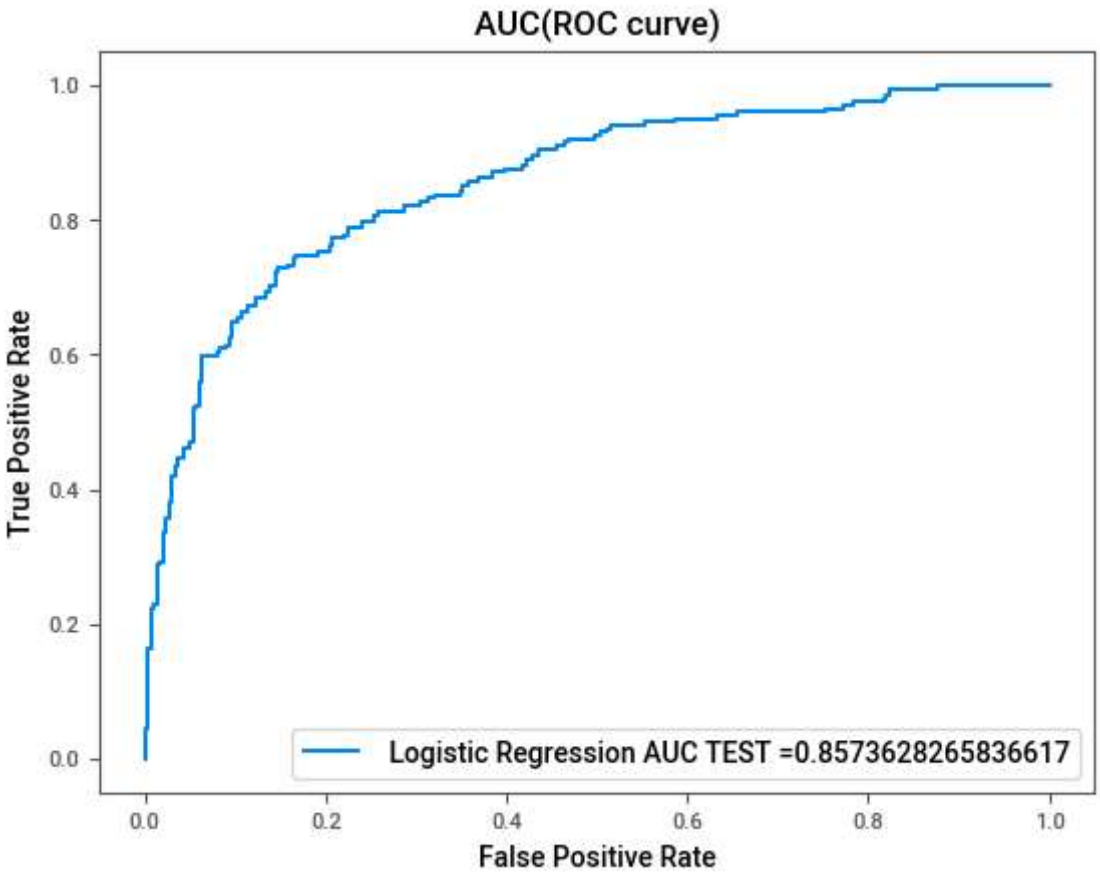
```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       299
           1       0.82      0.64      0.72       202

    accuracy                           0.80       501
   macro avg       0.80      0.77      0.78       501
weighted avg       0.80      0.80      0.79       501
```

In [26]:
```
y_pred_proba_lr = lr.predict_proba(X_test_scaled)[::,1]
fpr_lr, tpr_lr, _ = metrics.roc_curve(y_test,  y_pred_proba_lr)
```

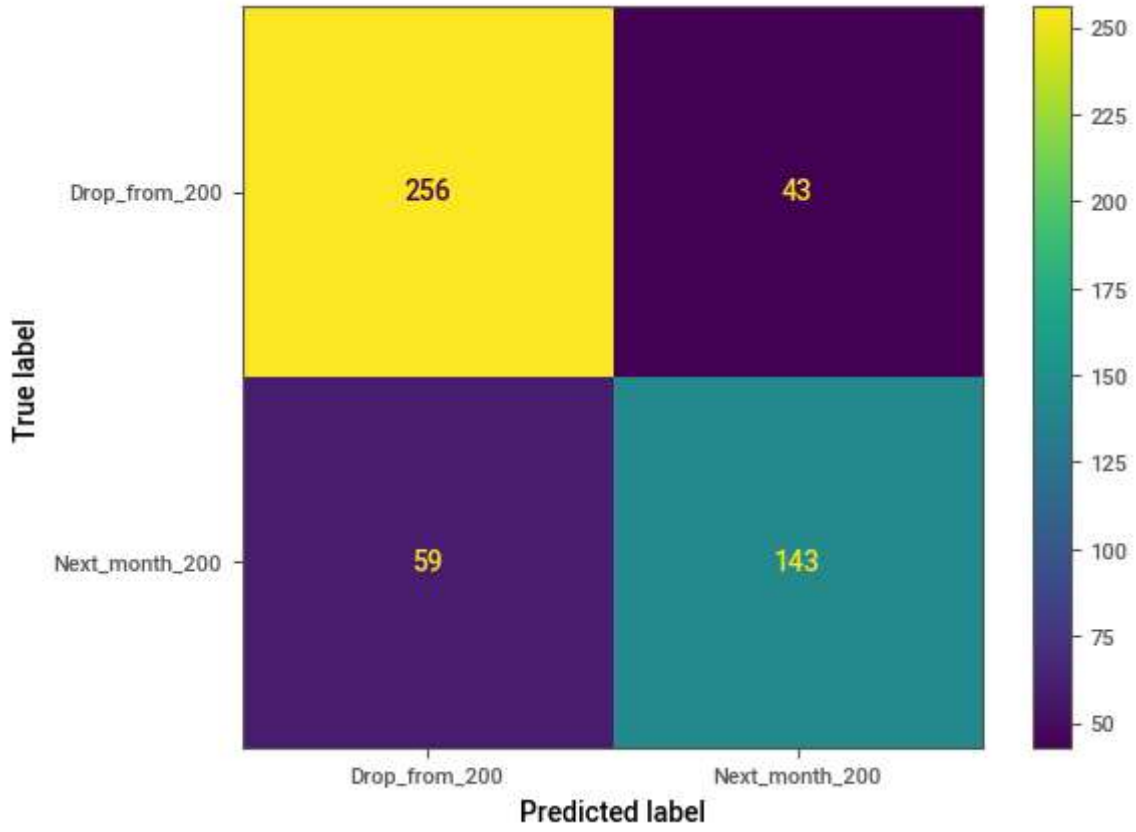
In [27]:
```
plt.plot(fpr_lr, tpr_lr, label=" Logistic Regression AUC TEST ="+str(auc(fpr_l
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC(ROC curve)")
plt.show()
```



In [28]:
```
y_pred_thresh = (lr.predict_proba(X_test_scaled)[:, 1] > .4).astype('float')
```

In [29]:
```
cm = confusion_matrix(y_test, y_pred_thresh, labels = lr.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = ['Drop_from_200', 'Next_m
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
ax.set_yticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
display_cm.plot(ax = ax)
```

Out[29]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a1f58cce
20>
```



In [30]:
```
print(classification_report(y_test, y_pred_thresh))
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       299
           1       0.77      0.71      0.74       202

    accuracy                           0.80       501
   macro avg       0.79      0.78      0.79       501
weighted avg       0.80      0.80      0.79       501
```

In [31]:
```
rf = RandomForestClassifier(random_state = 1701)
rf.fit(X_train_scaled, y_train)
```

Out[31]:
```
RandomForestClassifier(random_state=1701)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```
print("score on test: " + str(rf.score(X_test_scaled, y_test)))
print("score on train: "+ str(rf.score(X_train_scaled, y_train)))
```

```
score on test: 0.7684630738522954
score on train: 1.0
```

In [33]:
```
random_grid = {
 'max_depth': [3, 5, 7, 10, None],
 'min_samples_leaf': [5, 10, 50, 100],
 'n_estimators': [100],
 'random_state': [1701]}
```

In [34]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_gr
                               cv = 3, random_state=1701)
```

In [35]:
```
rf_random.fit(X_train_scaled, y_train)
```

Out[35]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1701),
                   n_iter=100,
                   param_distributions={'max_depth': [3, 5, 7, 10, None],
                                        'min_samples_leaf': [5, 10, 50, 100],
                                        'n_estimators': [100],
                                        'random_state': [1701]},
                   random_state=1701)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [36]:
```
print("score on test: " + str(rf_random.score(X_test_scaled, y_test)))
print("score on train: "+ str(rf_random.score(X_train_scaled, y_train)))
```

```
score on test: 0.7884231536926147
score on train: 0.8314457028647568
```

In [37]:
```
rf_random.best_params_
```

Out[37]:
```
{'random_state': 1701,
 'n_estimators': 100,
 'min_samples_leaf': 5,
 'max_depth': 7}
```

In [38]:
```
param_grid = {
 'max_depth': [3, 5, 7, 10, None],
 'min_samples_leaf': [5, 10, 50, 100],
 'n_estimators': [100]}
```

In [39]:
```
rf_grid = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 3)
```

In [40]:
```
rf_grid.fit(X_train_scaled, y_train)
```

Out[40]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1701),
             param_grid={'max_depth': [3, 5, 7, 10, None],
                         'min_samples_leaf': [5, 10, 50, 100],
                         'n_estimators': [100]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [41]:
```
print("score on test: " + str(rf_grid.score(X_test_scaled, y_test)))
print("score on train: "+ str(rf_grid.score(X_train_scaled, y_train)))
```

```
score on test: 0.7884231536926147
score on train: 0.8314457028647568
```

In [42]:
```
rf_grid.best_params_
```

Out[42]:
```
{'max_depth': 7, 'min_samples_leaf': 5, 'n_estimators': 100}
```

In [43]:
```
rf_nest_2000 = RandomForestClassifier(max_depth = 7, min_samples_leaf = 5, n_e
```

In [44]:
```
rf_nest_2000.fit(X_train_scaled, y_train)
```

Out[44]:
```
RandomForestClassifier(max_depth=7, min_samples_leaf=5, n_estimators=2000,
                       random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [45]:
```
print("score on test: " + str(rf_nest_2000.score(X_test_scaled, y_test)))
print("score on train: "+ str(rf_nest_2000.score(X_train_scaled, y_train)))
```

```
score on test: 0.782435129740519
score on train: 0.8287808127914723
```

In [46]:
```
importances = rf_nest_2000.feature_importances_
```

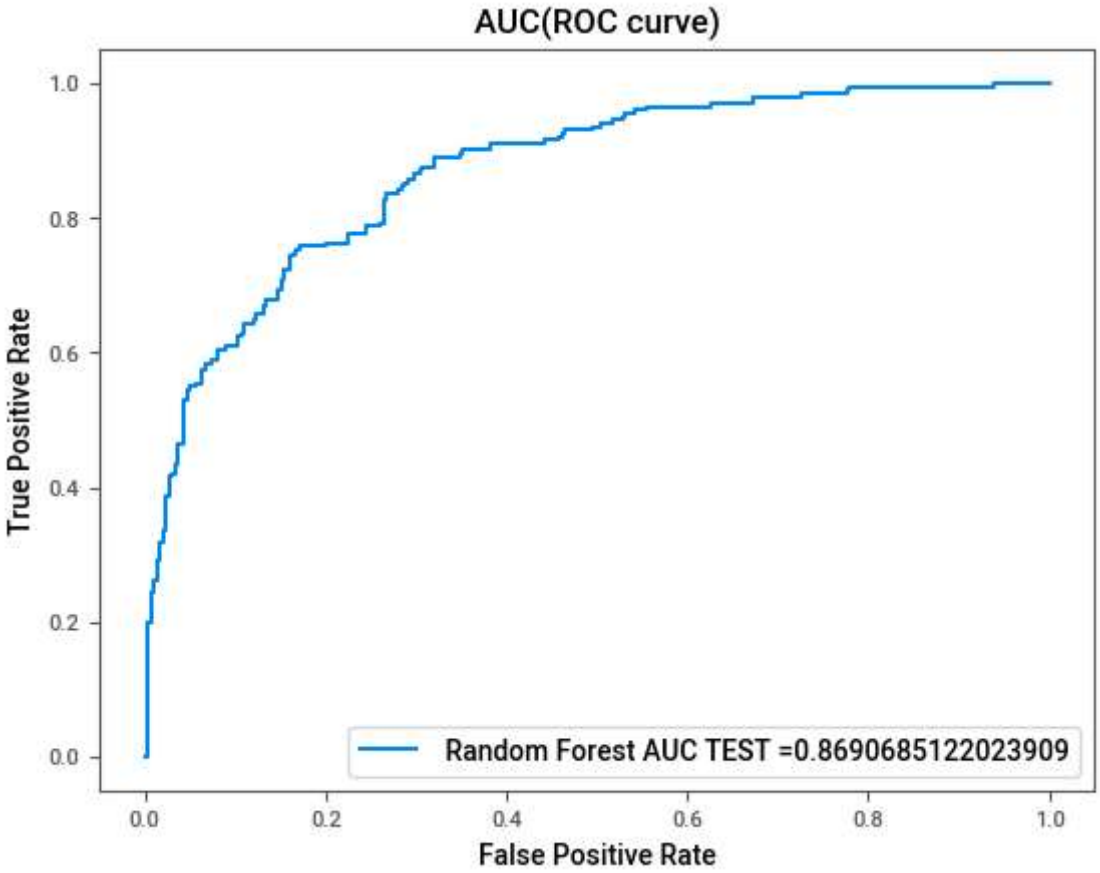
In [47]:
```
rf_importances = pd.Series(importances, index = X_test_scaled.columns)
rf_importances = rf_importances.sort_values(ascending = False)
```

In [48]:
```
rf_importances.head(10)
```

Out[48]:
```
Hours_streamed       0.189236
Avg_channels         0.135267
Jan_Debut_Month      0.126437
Hours_watched        0.110559
Streamers            0.109665
Avg_viewers          0.106297
Peak_channels        0.102155
Peak_viewers         0.096424
Avg_channel_ratio    0.023961
dtype: float64
```

In [49]:
```
y_pred_proba_rf = rf_nest_2000.predict_proba(X_test_scaled)[::,1]
fpr_rf, tpr_rf, _ = metrics.roc_curve(y_test,  y_pred_proba_rf)
```

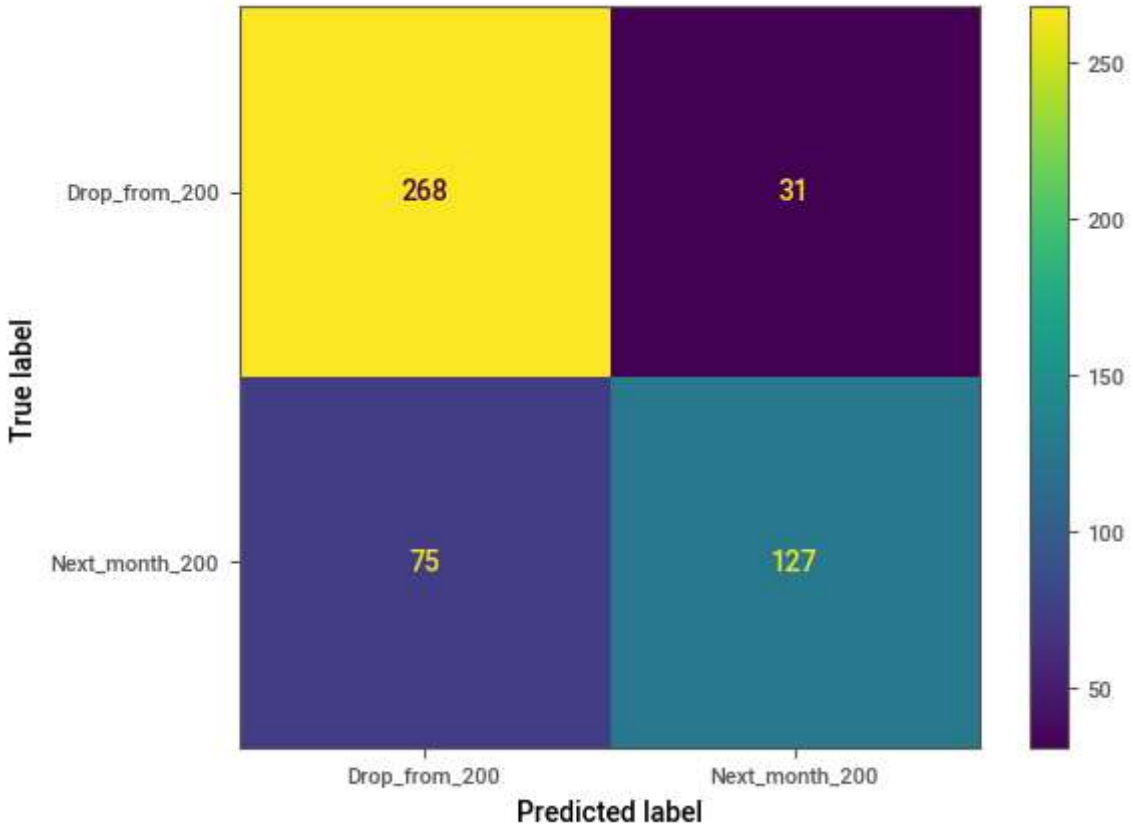
In [50]:
```
plt.plot(fpr_rf, tpr_rf, label=" Random Forest AUC TEST ="+str(auc(fpr_rf, tpr
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC(ROC curve)")
plt.show()
```



In [51]:
```
y_pred_test_rf = rf_grid.predict(X_test_scaled)
```

In [52]:
```
cm = confusion_matrix(y_test, y_pred_test_rf, labels = rf_grid.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = ['Drop_from_200', 'Next_m
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
ax.set_yticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
display_cm.plot(ax = ax)
```

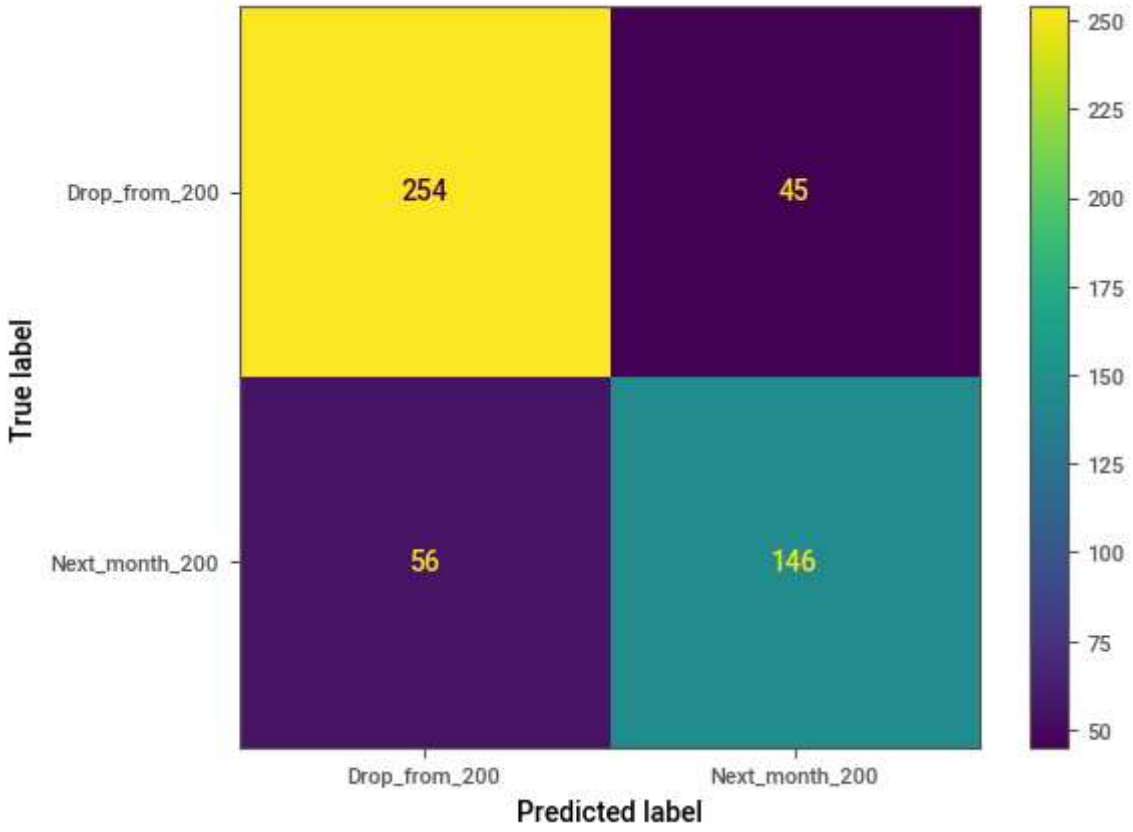
Out[52]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a1f5d17d
00>
```



In [53]:
```
y_pred_thresh = (rf_grid.predict_proba(X_test_scaled)[:, 1] > .44).astype('flo
```

In [54]:
```
cm = confusion_matrix(y_test, y_pred_thresh, labels = lr.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = ['Drop_from_200', 'Next_m
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
ax.set_yticklabels(labels = ['Drop_from_200', 'Next_month_200'], fontsize = 8)
display_cm.plot(ax = ax)
```

Out[54]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a1f72c7e
b0>
```



In [55]:
```
print(classification_report(y_test, y_pred_thresh))
```

```
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       299
           1       0.76      0.72      0.74       202

    accuracy                           0.80       501
   macro avg       0.79      0.79      0.79       501
weighted avg       0.80      0.80      0.80       501
```

In [56]:
```
twitch_df_X_3mth = twitch_df_X[twitch_df_X['Date'] < '2022-12-12']
```

In [57]:
```
twitch_df_X_3mth.tail()
```

Out[57]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	St
2020	Ixion	12	2022	1035038	9529	47595	142	
2021	Marvel's Midnight Suns	12	2022	1986127	54569	27336	290	
2022	Project: Playtime	12	2022	699293	23731	141061	264	
2023	Stalcraft	12	2022	1461144	29178	11735	113	
2024	The Callisto Protocol	12	2022	8703710	215971	267668	2598	

In [58]:
```
twitch_df_X_3mth.count()
```

Out[58]:
```
Game                 1922
Month                1922
Year                 1922
Hours_watched        1922
Hours_streamed       1922
Peak_viewers         1922
Peak_channels        1922
Streamers            1922
Avg_viewers          1922
Avg_channels         1922
Avg_channel_ratio    1922
Date                 1922
one_month_future     1922
three_month_future   1922
six_month_future     1922
Hours_watched_1mth   1922
Hours_watched_3mth   1922
Hours_watched_6mth   1922
Jan_Debut_Month      1922
Next_mth_200         1922
dtype: int64
```

In [59]:
```
twitch_df_X_3mth.reset_index(drop = True, inplace = True)
```

In [60]:
```
twitch_df_X.head()
```

Out[60]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Strea
0	7 Days to Die	1	2016	269681	12131	4405	44	
1	Agar.io	1	2016	255617	20705	4183	74	
2	Age of Empires	1	2016	248884	232	107455	18	
3	Alien: Isolation	1	2016	264294	11799	9590	42	
4	American Truck Simulator	1	2016	314055	724	43089	48	

In [61]:
```
x = len(twitch_df_X_3mth)
top200_list_3mth = []
for i in range(x):
    if twitch_df_X_3mth['Hours_watched_3mth'][i] != 0 and twitch_df_X_3mth['Ho
        top200_list_3mth.append(1)
    else:
        top200_list_3mth.append(0)
```

In [62]:
```
twitch_df_X_3mth['three_mth_200'] = top200_list_3mth
```

In [63]:
```
len(twitch_df_X_3mth[twitch_df_X_3mth['Hours_watched_3mth'] != 0])
```

Out[63]: 408

In [64]:
```
sum(top200_list_3mth)
```

Out[64]: 373

In [65]:
```
base_3mth = twitch_df_X_3mth.sort_values(by = 'Hours_watched_1mth', ascending
```

In [66]:
```
base_pred_3mth = [1]*408 + [0]*(1922-408)
```

In [67]:
```
base_3mth['prediction'] = base_pred_3mth
```

In [68]:
```
base_3mth.head(2)
```

Out[68]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Str
609	Lost Ark	1	2022	13020879	217772	68925	900	
76	League of Legends	1	2016	94377226	1362044	530270	2903	

2 rows × 22 columns

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(twitch_df_X_3mth.drop(colu


                                                    twitch_df_X_3mth.three_mth
                                                    random_state=1701)
```

In [70]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_gr
                               cv = 3, random_state=1701)
```

In [71]:
```
rf_random.fit(X_train, y_train)
```

Out[71]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1701),
                   n_iter=100,
                   param_distributions={'max_depth': [3, 5, 7, 10, None],
                                        'min_samples_leaf': [5, 10, 50, 100],
                                        'n_estimators': [100],
                                        'random_state': [1701]},
                   random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [72]:
```
print("score on test: " + str(rf_random.score(X_test, y_test)))
print("score on train: "+ str(rf_random.score(X_train, y_train)))
```

```
score on test: 0.9313929313929314
score on train: 0.9382373351839001
```

In [73]:
```
rf_random.best_params_
```

Out[73]:
```
{'random_state': 1701,
 'n_estimators': 100,
 'min_samples_leaf': 5,
 'max_depth': 7}
```

In [74]:
```
rf_3mth = RandomForestClassifier(max_depth = 7, min_samples_leaf = 5, n_estima
```

In [75]:
```
rf_3mth.fit(X_train, y_train)
```

Out[75]:
```
RandomForestClassifier(max_depth=7, min_samples_leaf=5, n_estimators=2000,
                       random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [76]:
```
'Baseline data for comparison in cell below'
```

Out[76]:
```
'Baseline data for comparison in cell below'
```

In [77]:
```
print(classification_report(base_3mth['three_mth_200'], base_3mth['prediction'
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      1549
           1       0.55      0.60      0.57       373

    accuracy                           0.83      1922
   macro avg       0.72      0.74      0.73      1922
weighted avg       0.83      0.83      0.83      1922
```

In [78]:
```
y_pred_test_3mth = rf_3mth.predict(X_test)
```

In [79]:
```
'Random Forest Model 3 month, assuming knowledge of hours watched 1 monht afte
```

Out[79]:
```
'Random Forest Model 3 month, assuming knowledge of hours watched 1 monht aft
er debut'
```

In [80]:
```
print(classification_report(y_test, y_pred_test_3mth))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       388
           1       0.84      0.75      0.80        93

    accuracy                           0.93       481
   macro avg       0.89      0.86      0.87       481
weighted avg       0.92      0.93      0.92       481
```

In [81]:
```
importances = rf_3mth.feature_importances_
```

In [82]:
```
rf_importances = pd.Series(importances, index = X_test.columns)
rf_importances = rf_importances.sort_values(ascending = False)
```

In [83]:
```
rf_importances
```

Out[83]:
```
Hours_watched_1mth    0.371756
Jan_Debut_Month       0.230367
Hours_streamed        0.090541
Avg_channels          0.067230
Peak_viewers          0.058768
Streamers             0.042660
Peak_channels         0.040071
Hours_watched         0.040036
Avg_viewers           0.037139
Avg_channel_ratio     0.021432
dtype: float64
```

In [84]:
```
twitch_df_X_6mth = twitch_df_X[twitch_df_X['Date'] < '2022-09-01']
```

In [85]:
```
twitch_df_X_6mth.reset_index(drop = True, inplace = True)
```

In [86]:
```
x = len(twitch_df_X_6mth)
top200_list_6mth = []
for i in range(x):
    if twitch_df_X_6mth['Hours_watched_3mth'][i] != 0:
        top200_list_6mth.append(1)
    else:
        top200_list_6mth.append(0)
```

In [87]:
```
twitch_df_X_6mth['six_mth_200'] = top200_list_6mth
```

In [88]:
```
base_6mth = twitch_df_X_6mth.sort_values('Hours_watched_3mth', ascending = Fal
```

In [89]:
```
x = sum(top200_list_6mth)
x
```

Out[89]:
```
407
```

In [90]:
```
y = len(top200_list_6mth) - x
y
```

Out[90]:
```
1458
```

In [91]:
```
base_pred_6mth = [1]*x + [0]*y
```

In [92]:
```
base_6mth['prediction'] = base_pred_6mth
```

In [93]:
```
'''Both baseline and model make perfect predictions at 6 months from debut, as
watched 3 months from debut'''
```

Out[93]:
```
'Both baseline and model make perfect predictions at 6 months from debut, ass
uming knowledge of hours \nwatched 3 months from debut'
```

In [94]:
```
print(classification_report(base_6mth['six_mth_200'], base_6mth['prediction'])
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1458
           1       1.00      1.00      1.00       407

    accuracy                           1.00      1865
   macro avg       1.00      1.00      1.00      1865
weighted avg       1.00      1.00      1.00      1865
```

In [95]:
```
X_train, X_test, y_train, y_test = train_test_split(twitch_df_X_6mth.drop(colu
                                                                            '
                                                    twitch_df_X_6mth.six_mth_2
                                                    random_state=1701)
```

In [96]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_gr
                               cv = 3, random_state=1701)
```

In [97]:
```
rf_random.fit(X_train, y_train)
```

Out[97]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1701),
                   n_iter=100,
                   param_distributions={'max_depth': [3, 5, 7, 10, None],
                                        'min_samples_leaf': [5, 10, 50, 100],
                                        'n_estimators': [100],
                                        'random_state': [1701]},
                   random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [98]:
```
print("score on test: " + str(rf_random.score(X_test, y_test)))
print("score on train: "+ str(rf_random.score(X_train, y_train)))
```

```
score on test: 1.0
score on train: 1.0
```

In [99]:
```
'''Data sets starts at Jan 2016, therefore I wanted to test any oddities of ga
I removed the data from January of 2016 and ran a model to test accuracy. Some
due to loss of data. But signficant difference would be worrying'''
```

Out[99]:
```
'Data sets starts at Jan 2016, therefore I wanted to test any oddities of gam
es debuting in Jan 2016. \nI removed the data from January of 2016 and ran a
model to test accuracy. Some decrease in accuracy is expected\ndue to loss of
data. But signficant difference would be worrying'
```

In [100]:
```
no_Jan2016 = twitch_df_X[twitch_df_X['Date'] != '2016-01-01']
no_Jan2016.head(2)
```

Out[100]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels
200	ARK: Survival Evolved	1	2017	2167646	192501	18756	483
201	ASTRONEER	1	2017	761112	21225	29721	72

In [101]:
```
X_train, X_test, y_train, y_test = train_test_split(no_Jan2016.drop(columns=['



                                                     no_Jan2016.Next_mth_200, t
                                                     random_state=1701)
```

In [102]:
```
rf_noJan2016 = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 3)
```

In [103]:
```
rf_noJan2016.fit(X_train, y_train)
```

Out[103]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1701),
             param_grid={'max_depth': [3, 5, 7, 10, None],
                         'min_samples_leaf': [5, 10, 50, 100],
                         'n_estimators': [100]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [104]:
```
rf_noJan2016.best_params_
```

Out[104]:
```
{'max_depth': 7, 'min_samples_leaf': 10, 'n_estimators': 100}
```

In [105]:
```
rf_noJan2016 = RandomForestClassifier(max_depth = 7, min_samples_leaf = 10, n_
```

In [106]:
```
rf_noJan2016.fit(X_train, y_train)
```

Out[106]:
```
RandomForestClassifier(max_depth=7, min_samples_leaf=10, random_state=1701)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [107]:
```
'''The accuracy score below is enough to determine that there is no great oddi
as a debut month, despite this likely being not a true debut month for many of
```

Out[107]:
```
'The accuracy score below is enough to determine that there is no great oddit
ies in the data resulting from using Jan 2016\nas a debut month, despite this
likely being not a true debut month for many of the games on this list'
```

In [108]:
```
print("score on test: " + str(rf_noJan2016.score(X_test, y_test)))
print("score on train: "+ str(rf_noJan2016.score(X_train, y_train)))
```

```
score on test: 0.753880266075388
score on train: 0.8016284233900814
```

In [109]:
```
'''I do not have data from this dataset before they entered the top 200. Howev
from the debut month to the next. I have already attempted to predict the zero
games that were in the top 200 2 months in a row.'''
```

Out[109]:
```
'I do not have data from this dataset before they entered the top 200. Howeve
r, I can attempt to predict the hours watched \nfrom the debut month to the n
ext. I have already attempted to predict the zeroes. So this will attempt the
hours watched for\ngames that were in the top 200 2 months in a row.'
```

In [110]:
```
noz_df = twitch_df_X[twitch_df_X['Hours_watched_1mth'] != 0]
```

In [111]:
```
noz_df.head(2)
```

Out[111]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Stream
0	7 Days to Die	1	2016	269681	12131	4405	44	1
1	Agar.io	1	2016	255617	20705	4183	74	4

In [112]:
```
def regression_metrics(y_true, y_pred):
    meanAbErr = metrics.mean_absolute_error(y_true, y_pred)
    meanSqErr = metrics.mean_squared_error(y_true, y_pred)
    rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    mape = mean_absolute_percentage_error(y_true, y_pred)
    print('R squared:', r2)
    print('Mean Absolute Error:', meanAbErr)
    print('Mean Square Error:', meanSqErr)
    print('Root Mean Square Error:', rootMeanSqErr)
    print('Mean Absolute Percentage Error:', mape)
```

In [113]:
```
base_noz = noz_df
```

In [114]:
```
base_noz['1mth_diff'] = base_noz['Hours_watched_1mth'] - base_noz['Hours_watch
```

In [115]:
```
base_noz['1mth_diff'].median()
```

Out[115]: -154576.0

In [116]:
```
base_noz['prediction'] = base_noz['Hours_watched'] + base_noz['1mth_diff'].med
```

In [117]:
```
x = noz_df.drop(columns=['Game', 'Month', 'Year', 'Date', 'one_month_future',



X_train, X_test, y_train, y_test = train_test_split(x,
                                                    noz_df.Hours_watched_1mth,
```

In [118]:
```
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.c
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns = X_test.columr
X_train_scaled.head()
```

Out[118]:

	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Streamers	Avg_viewers	Avg
0	-0.256115	-0.319352	-0.272180	-0.330493	-0.403284	-0.256848	
1	-0.274355	-0.294672	0.224038	-0.317483	-0.339148	-0.270921	
2	-0.297519	-0.115383	-0.270431	0.295264	-0.201626	-0.297627	
3	-0.121909	-0.240355	0.615694	-0.237475	-0.194777	-0.124755	
4	-0.286662	-0.053396	-0.602511	-0.247882	-0.144923	-0.286944	

In [119]:
```
mlr = LinearRegression()
```

In [120]:
```
mlr.fit(X_train_scaled, y_train)
```

Out[120]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [121]:
```
y_pred_mlr= mlr.predict(X_test_scaled)
```

In [122]:
```
mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr
mlr_diff.head()
```

Out[122]:

	Actual value	Predicted value
1034	3650194	1.284599e+06
588	660872	1.815768e+06
177	374856	1.011052e+06
303	166134	9.005599e+05
77	222387	-3.530944e+05

In [123]:
```
mlr_diff[mlr_diff['Actual value'] == 166134]
```

Out[123]:

	Actual value	Predicted value
303	166134	900559.924247

In [124]:
```
'LINEAR REGRESSION ERROR 1MTH'
```

Out[124]:
```
'LINEAR REGRESSION ERROR 1MTH'
```

In [125]:
```
regression_metrics(y_test, y_pred_mlr)
```

```
R squared: 0.8614427353212233
Mean Absolute Error: 1739549.4728041925
Mean Square Error: 13220153993562.62
Root Mean Square Error: 3635952.969107634
Mean Absolute Percentage Error: 1.8657147562948042
```

In [126]:
```
rfr = RandomForestRegressor(random_state=1701)
```

In [127]:
```
rfr.fit(X_train, y_train)
```

Out[127]:
```
RandomForestRegressor(random_state=1701)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [128]:
```
'BASELINE PREDICTION MEDIAN ADJUSTED 1MTH'
```

Out[128]: 'BASELINE PREDICTION MEDIAN ADJUSTED 1MTH'

In [129]:
```
regression_metrics(base_noz['Hours_watched_1mth'], base_noz['prediction'])
```

```
R squared: 0.37027697052803776
Mean Absolute Error: 1688917.0914709517
Mean Square Error: 43763070775200.21
Root Mean Square Error: 6615366.261606398
Mean Absolute Percentage Error: 1.1493906419960174
```

In [130]:
```
'BASELINE PREDICTION HOURS WATCHED 1MTH'
```

Out[130]: 'BASELINE PREDICTION HOURS WATCHED 1MTH'

In [131]:
```
regression_metrics(base_noz['Hours_watched_1mth'], base_noz['Hours_watched'])
```

```
R squared: 0.3677506871227991
Mean Absolute Error: 1715592.5624227442
Mean Square Error: 43938636721318.42
Root Mean Square Error: 6628622.535739867
Mean Absolute Percentage Error: 1.2059516193213438
```

In [132]:
```
y_pred_rfr= rfr.predict(X_test)
```

In [133]:
```
'RANDOM FOREST REGRESSOR NO HYPERPARAMETER TUNING 1MTH'
```

Out[133]: 'RANDOM FOREST REGRESSOR NO HYPERPARAMETER TUNING 1MTH'

In [134]:
```
regression_metrics(y_test, y_pred_rfr)
```

```
R squared: 0.5703755626753286
Mean Absolute Error: 1705261.5656157632
Mean Square Error: 40991724497429.59
Root Mean Square Error: 6402477.996637676
Mean Absolute Percentage Error: 1.0507723783818461
```

In [135]:
```
random_grid = {'max_depth': [3, 5, 7, 10, None],
 'min_samples_leaf': [5, 10, 50, 100],
 'n_estimators': [100]}
```

In [136]:
```
rfr = RandomForestRegressor(random_state = 1701)
rfr_random = RandomizedSearchCV(estimator = rfr, param_distributions = random_
```

In [137]:
```
rfr_random.fit(X_train, y_train)
```

Out[137]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(random_state=1701),
                   n_iter=100,
                   param_distributions={'max_depth': [3, 5, 7, 10, None],
                                        'min_samples_leaf': [5, 10, 50, 100],
                                        'n_estimators': [100]},
                   random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [138]:
```
rfr_random.best_params_
```

Out[138]:
```
{'n_estimators': 100, 'min_samples_leaf': 50, 'max_depth': 5}
```

In [139]:
```
'Both Random Search and Grid Search are worse than the base model by a lot'
```

Out[139]:
```
'Both Random Search and Grid Search are worse than the base model by a lot'
```

In [140]:
```
y_pred_rfr_rand = rfr_random.predict(X_test)
```

In [141]:
```
'RANDOM SEARCH RANDOM FOREST REGRESSOR 1 MONTH'
```

Out[141]:
```
'RANDOM SEARCH RANDOM FOREST REGRESSOR 1 MONTH'
```

In [142]:
```
regression_metrics(y_test, y_pred_rfr_rand)
```

```
R squared: 0.23314171444331955
Mean Absolute Error: 2653188.873840058
Mean Square Error: 73168192586668.47
Root Mean Square Error: 8553840.809055805
Mean Absolute Percentage Error: 1.382616976585178
```

In [143]:
```
rfr = RandomForestRegressor(random_state = 1701)
rfr_grid = GridSearchCV(estimator = rfr, param_grid = random_grid, cv = 3)
```

In [144]:
```
rfr_grid.fit(X_train, y_train)
```

Out[144]:
```
GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=1701),
             param_grid={'max_depth': [3, 5, 7, 10, None],
                         'min_samples_leaf': [5, 10, 50, 100],
                         'n_estimators': [100]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [145]:
```
rfr_grid.best_params_
```

Out[145]:
```
{'max_depth': 5, 'min_samples_leaf': 50, 'n_estimators': 100}
```

In [146]:
```
rfr_best = RandomForestRegressor(max_depth = 5, min_samples_leaf = 50, n_estim
```

In [147]:
```
rfr_best.fit(X_train, y_train)
```

Out[147]:
```
RandomForestRegressor(max_depth=5, min_samples_leaf=50, n_estimators=2000,
                      random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [148]:
```
y_pred = rfr_best.predict(X_test)
```

In [149]:
```
'GRID SEARCH RANDOM FOREST REGRESSOR 1MTH'
```

Out[149]:
```
'GRID SEARCH RANDOM FOREST REGRESSOR 1MTH'
```

In [150]:
```
regression_metrics(y_test, y_pred)
```

```
R squared: 0.23012074742000344
Mean Absolute Error: 2638048.6302234423
Mean Square Error: 73456431888666.11
Root Mean Square Error: 8570672.779231869
Mean Absolute Percentage Error: 1.3689547937881816
```

In [151]:
```
'''Note: there are 35 games that dropped out of the top 200 after its debut mo
third month post-debut. These were removed from the below regression, despite
This is because these would have a less significant detrimental impact on the
to the regression models'''
```

Out[151]:
```
'Note: there are 35 games that dropped out of the top 200 after its debut mon
th and re-entered the top 200 by the \nthird month post-debut. These were rem
oved from the below regression, despite being left in for the classification
models. \nThis is because these would have a less significant detrimental imp
act on the classification models as compared \nto the regression models'
```

In [152]:
```
noz_df_3mth = noz_df[noz_df['Hours_watched_3mth'] != 0]
```

In [153]:
```
noz_df_3mth.head(2)
```

Out[153]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Stream
0	7 Days to Die	1	2016	269681	12131	4405	44	1:
1	Agar.io	1	2016	255617	20705	4183	74	4

2 rows × 22 columns

In [154]:
```
x = noz_df_3mth.drop(columns=['Game', 'Month', 'Year', 'Date', 'one_month_futu



X_train, X_test, y_train, y_test = train_test_split(x,
                                                    noz_df_3mth.Hours_watched_
```

In [155]:
```
rfr_3mth = RandomForestRegressor(random_state=1701)
```

In [156]:
```
rfr_3mth.fit(X_train, y_train)
```

Out[156]:
```
RandomForestRegressor(random_state=1701)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [157]:
```
y_pred_3mth = rfr_3mth.predict(X_test)
```

In [158]:
```
'RANDOM FOREST REGRESSOR NO HYPERPARAMETER TUNING 3MTH'
```

Out[158]:
```
'RANDOM FOREST REGRESSOR NO HYPERPARAMETER TUNING 3MTH'
```

In [159]:
```
regression_metrics(y_test, y_pred_3mth)
```

```
R squared: 0.7135702955279529
Mean Absolute Error: 1889323.789361702
Mean Square Error: 37345639982178.02
Root Mean Square Error: 6111107.917732923
Mean Absolute Percentage Error: 1.0682930769804502
```

In [160]:
```
rfr_3mth = RandomForestRegressor(random_state = 1701)
rfr_grid_3mth = GridSearchCV(estimator = rfr_3mth, param_grid = random_grid, c
```

In [161]:
```
rfr_grid_3mth.fit(X_train, y_train)
```

Out[161]:
```
GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=1701),
             param_grid={'max_depth': [3, 5, 7, 10, None],
                         'min_samples_leaf': [5, 10, 50, 100],
                         'n_estimators': [100]})
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [162]:
```
rfr_grid_3mth.best_params_
```

Out[162]:
```
{'max_depth': None, 'min_samples_leaf': 5, 'n_estimators': 100}
```

In [163]:
```
rfr_3mth_best = RandomForestRegressor(max_depth = None, min_samples_leaf = 5,
```

In [164]:
```
rfr_3mth_best.fit(X_train, y_train)
```

Out[164]:
```
RandomForestRegressor(min_samples_leaf=5, n_estimators=2000, random_state=170
1)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [165]:
```
y_pred_3mth = rfr_3mth_best.predict(X_test)
```

In [166]:
```
'GRID SEARCH RANDOM FOREST REGRESSOR 3MTH (Hypertuned)'
```

Out[166]:
```
'GRID SEARCH RANDOM FOREST REGRESSOR 3MTH (Hypertuned)'
```

In [167]:
```
regression_metrics(y_test, y_pred_3mth)
```

```
R squared: 0.8531265620397721
Mean Absolute Error: 1604665.1009233268
Mean Square Error: 19149838342072.953
Root Mean Square Error: 4376052.826700444
Mean Absolute Percentage Error: 1.202212474047224
```

In [168]:
```
base_noz_3mth = noz_df_3mth
```

In [169]:
```
base_noz_3mth['3mth_diff'] = base_noz['Hours_watched_3mth'] - base_noz['Hours_
```

In [170]:
```
base_noz_3mth['3mth_diff'].median()
```

Out[170]:
```
-41334.0
```

In [171]:
```
base_noz_3mth['prediction'] = base_noz_3mth['Hours_watched'] + base_noz_3mth['
```

In [172]:
```
'BASELINE PREDICTION MEDIAN BASED'
```

Out[172]:
```
'BASELINE PREDICTION MEDIAN BASED'
```

In [173]:
```
regression_metrics(base_noz_3mth['Hours_watched_3mth'], base_noz_3mth['predict
```

```
R squared: 0.5578602703505053
Mean Absolute Error: 2348520.4718498657
Mean Square Error: 65311764264878.28
Root Mean Square Error: 8081569.41842847
Mean Absolute Percentage Error: 1.4938274267888494
```

In [174]:
```
'BASELINE PREDICTION 1MTH HOURS WATCHED'
```

Out[174]:
```
'BASELINE PREDICTION 1MTH HOURS WATCHED'
```

In [175]:
```
regression_metrics(base_noz_3mth['Hours_watched_3mth'], base_noz_3mth['Hours_w
```

```
R squared: 0.4998744001790798
Mean Absolute Error: 1981199.4798927614
Mean Square Error: 73877290566557.25
Root Mean Square Error: 8595189.966868518
Mean Absolute Percentage Error: 1.2249099401553
```

In [176]:
```
noz_df_6mth = noz_df_3mth[noz_df_3mth['Hours_watched_6mth'] != 0]
```

In [177]:
```
x = noz_df_6mth.drop(columns=['Game', 'Month', 'Year', 'Date', 'one_month_futu



X_train, X_test, y_train, y_test = train_test_split(x,
                                                    noz_df_6mth.Hours_watched_
```

In [178]:
```
rfr_6mth = RandomForestRegressor(random_state = 1701)
```

In [179]:
```
rfr_6mth.fit(X_train, y_train)
```

Out[179]:
```
RandomForestRegressor(random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [180]:
```
y_pred_6mth = rfr_6mth.predict(X_test)
```

In [181]:
```
'RANDOM FOREST REGRESSOR NO HYPERPARAMETER TUNING 6MTH'
```

Out[181]: 'RANDOM FOREST REGRESSOR NO HYPERPARAMETER TUNING 6MTH'

In [182]:
```
regression_metrics(y_test, y_pred_6mth)
```

```
R squared: 0.5772557124359676
Mean Absolute Error: 3231574.599848485
Mean Square Error: 82453434480670.25
Root Mean Square Error: 9080387.353008144
Mean Absolute Percentage Error: 0.648817838299254
```

In [183]:
```
base_noz_6mth = noz_df_6mth
```

In [184]:
```
base_noz_6mth['6mth_diff'] = base_noz['Hours_watched_6mth'] - base_noz['Hours_
```

In [185]:
```
base_noz_6mth['prediction'] = base_noz_6mth['Hours_watched'] + base_noz_3mth['
```

In [186]:
```
'BASELINE PREDICTION MEDIAN BASED'
```

Out[186]: 'BASELINE PREDICTION MEDIAN BASED'

In [187]:
```
regression_metrics(base_noz_6mth['Hours_watched_6mth'], base_noz_6mth['predict
```

```
R squared: 0.477595472076515
Mean Absolute Error: 3283737.0344827585
Mean Square Error: 111711147978406.83
Root Mean Square Error: 10569349.458618863
Mean Absolute Percentage Error: 1.2198957765530005
```

In [188]:
```
'BASELINE PREDICTION HOURS WATCHED 3MTH'
```

Out[188]: 'BASELINE PREDICTION HOURS WATCHED 3MTH'

In [189]:
```
regression_metrics(base_noz_6mth['Hours_watched_6mth'], base_noz_6mth['Hours_w
```

```
R squared: 0.9034256925732245
Mean Absolute Error: 1676340.5057471264
Mean Square Error: 20651480167577.76
Root Mean Square Error: 4544389.966494706
Mean Absolute Percentage Error: 0.5519367396774476
```

In [190]:
```
rfr_6mth = RandomForestRegressor(random_state = 1701)
rfr_grid_6mth = GridSearchCV(estimator = rfr_6mth, param_grid = random_grid, c
```

In [191]:
```
rfr_grid_6mth.fit(X_train, y_train)
```

Out[191]:
```
GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=1701),
             param_grid={'max_depth': [3, 5, 7, 10, None],
                         'min_samples_leaf': [5, 10, 50, 100],
                         'n_estimators': [100]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [192]:
```
rfr_grid_6mth.best_params_
```

Out[192]:
```
{'max_depth': 5, 'min_samples_leaf': 5, 'n_estimators': 100}
```

In [193]:
```
rfr_6mth_best = RandomForestRegressor(max_depth = 5, min_samples_leaf = 5, n_e
```

In [194]:
```
rfr_6mth_best.fit(X_train, y_train)
```

Out[194]:
```
RandomForestRegressor(max_depth=5, min_samples_leaf=5, n_estimators=2000,
                      random_state=1701)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [195]:
```
y_pred_6mth = rfr_6mth_best.predict(X_test)
```

In [196]:
```
'GRID SEARCH RANDOM FOREST REGRESSOR 6MTH'
```

Out[196]:
```
'GRID SEARCH RANDOM FOREST REGRESSOR 6MTH'
```

In [197]:
```
regression_metrics(y_test, y_pred_6mth)
```

```
R squared: 0.26855223834199504
Mean Absolute Error: 3751720.120375608
Mean Square Error: 142663974099865.53
Root Mean Square Error: 11944202.53092962
Mean Absolute Percentage Error: 0.6875374393358996
```

In [198]:
```
glob_df = pd.read_csv('Twitch_global_data.csv')
```

In [199]:
```
glob_df.head(2)
```

Out[199]:

	year	Month	Hours_watched	Avg_viewers	Peak_viewers	Streams	Avg_channels	Games_st
0	2016	1	480241904	646355	1275257	7701675	20076	
1	2016	2	441859897	635769	1308032	7038520	20427	

In [200]:
```
noz_glob = noz_df
```

In [201]:
```
noz_df.reset_index(inplace = True, drop = True)
```

In [202]:
```
x = len(noz_df)
y = len(glob_df)
tot_hours = []
for i in range (x):
    for j in range(y):
        if noz_df['Month'][i] == glob_df['Month'][j] and noz_df['Year'][i] ==
            tot_hours.append(glob_df['Hours_watched'][j])
```

In [203]:
```
noz_glob['Twitch_tot_hrs'] = tot_hours
```

In [204]:
```
noz_glob['Percent_total_hours'] = noz_glob['Hours_watched']/noz_glob['Twitch_t
```

In [205]:
```
noz_glob.head(2)
```

Out[205]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Stream
0	7 Days to Die	1	2016	269681	12131	4405	44	1:
1	Agar.io	1	2016	255617	20705	4183	74	4

2 rows × 24 columns

In [206]:
```
df = list(glob_df.year.astype(str) + '/' + glob_df.Month.astype(str) + '/01')
glob_df['Date'] = df
glob_df['Date']= pd.to_datetime(glob_df['Date'])
```

In [207]:
```
glob_df.head(2)
```

Out[207]:

	year	Month	Hours_watched	Avg_viewers	Peak_viewers	Streams	Avg_channels	Games_st
0	2016	1	480241904	646355	1275257	7701675	20076	
1	2016	2	441859897	635769	1308032	7038520	20427	

In [208]:
```
tot_hrs_1mth = []
for i in range(x):
    for j in range(y):
        if noz_glob['one_month_future'][i] == glob_df['Date'][j]:
            tot_hrs_1mth.append(glob_df['Hours_watched'][j])
```

In [209]:
```
noz_glob['1mth_tot_hrs'] = tot_hrs_1mth
```

In [210]:
```
noz_glob.head(2)
```

Out[210]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Stream
0	7 Days to Die	1	2016	269681	12131	4405	44	1:
1	Agar.io	1	2016	255617	20705	4183	74	4

2 rows × 25 columns

In [211]:
```
noz_glob['%_hrs_watched_1mth'] = noz_glob['Hours_watched_1mth']/noz_glob['1mth
```

In [212]:
```
noz_glob.head(2)
```

Out[212]:

	Game	Month	Year	Hours_watched	Hours_streamed	Peak_viewers	Peak_channels	Stream
0	7 Days to Die	1	2016	269681	12131	4405	44	1:
1	Agar.io	1	2016	255617	20705	4183	74	4

2 rows × 26 columns

In [213]:
```
'Predicting % of total hours watched on Twitch by game one month after debut'
```

Out[213]:
```
'Predicting % of total hours watched on Twitch by game one month after debut'
```

In [214]:
```
X_train, X_test, y_train, y_test = train_test_split(noz_glob.drop(columns=['Ga
                                                                            noz_glob['%_hrs_watched_1m
```

In [215]:
```
rf = RandomForestRegressor(random_state = 1701)
```

In [216]:
```
best_score = 0
best_grid = 0
for i in ParameterGrid(random_grid):
    rf.set_params(**i)
    rf.fit(X_train,y_train)
    y_pred = rf.predict(X_test)
    # save if best
    if r2_score(y_test, y_pred) > best_score:
        best_score = r2_score(y_test, y_pred)
        best_grid = i

print ("r2: %0.5f" % best_score)
print ("Grid:", best_grid)
```

```
r2: 0.58217
Grid: {'max_depth': None, 'min_samples_leaf': 5, 'n_estimators': 100}
```

In [217]:
```
best_rf = RandomForestRegressor(max_depth = None, min_samples_leaf = 5, n_esti
```

In [218]:
```
best_rf.fit(X_train, y_train)
```

Out[218]:
```
RandomForestRegressor(min_samples_leaf=5, n_estimators=2000, random_state=170
1)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [219]:
```
y_train_pred = best_rf.predict(X_train)
y_test_pred = best_rf.predict(X_test)
```

In [220]:
```
print('r2: ', r2_score(y_train, y_train_pred))
print('r2: ', r2_score(y_test, y_test_pred))
```

```
r2:  0.9068067521550346
r2:  0.6053586010261639
```

In [221]:
```
best_rf_100 = RandomForestRegressor(max_depth = None, min_samples_leaf = 5, n_
```

In [222]:
```
y_train_pred_100 = best_rf.predict(X_train)
y_test_pred_100 = best_rf.predict(X_test)
```

In [223]:
```
'This is only a slightly better r2 than when I used a base model attempting to
```

Out[223]:
```
'This is only a slightly better r2 than when I used a base model attempting t
o predict hours watched 1 month from debut'
```

In [224]:
```
print('r2: Train ', r2_score(y_train, y_train_pred_100))
print('r2: Test', r2_score(y_test, y_test_pred_100))
```

```
r2: Train  0.9068067521550346
r2: Test 0.6053586010261639
```

In [225]:
```
'Predicting % of total hours watched on Twitch by game three months after debu
```

Out[225]:
```
'Predicting % of total hours watched on Twitch by game three months after deb
ut.'
```

In [226]:
```
noz_glob_3mth = (noz_glob[noz_glob['Hours_watched_3mth'] != 0])
noz_glob_3mth.reset_index(drop = True, inplace = True)
```

In [227]:
```
x = len(noz_glob_3mth)
tot_hrs_3mth = []
for i in range(x):
    for j in range(y):
        if noz_glob_3mth['three_month_future'][i] == glob_df['Date'][j]:
            tot_hrs_3mth.append(glob_df['Hours_watched'][j])
```

In [228]:
```
noz_glob_3mth['3mth_tot_hrs'] = tot_hrs_3mth
```

In [229]:
```
noz_glob_3mth['%_hrs_watched_3mth'] = noz_glob_3mth['Hours_watched_3mth']/noz_
```

In [230]:
```
X_train, X_test, y_train, y_test = train_test_split(noz_glob_3mth.drop(columns
                                                                       noz_glob_3mth['%_hrs_watch
```

In [231]:
```
rf_3mth = RandomForestRegressor(random_state = 1701)
```

In [232]:
```
'''This is a worse score than the hypertuned model and given there was no huge
adds little value'''
```

Out[232]:
```
'This is a worse score than the hypertuned model and given there was no huge
issues with the 3 month variant this also \nadds little value'
```

In [233]:
```
best_score = 0
best_grid = 0
for i in ParameterGrid(random_grid):
    rf_3mth.set_params(**i)
    rf_3mth.fit(X_train,y_train)
    y_pred = rf_3mth.predict(X_test)
    # save if best
    if r2_score(y_test, y_pred) > best_score:
        best_score = r2_score(y_test, y_pred)
        best_grid = i

print ("r2: %0.5f" % best_score)
print ("Grid:", best_grid)
```

```
r2: 0.76102
Grid: {'max_depth': 10, 'min_samples_leaf': 5, 'n_estimators': 100}
```

In [234]:
```
noz_glob_6mth = (noz_glob_3mth[noz_glob_3mth['Hours_watched_6mth'] != 0])
noz_glob_6mth.reset_index(drop = True, inplace = True)
```

In [235]:
```
x = len(noz_glob_6mth)
tot_hrs_6mth = []
for i in range(x):
    for j in range(y):
        if noz_glob_6mth['six_month_future'][i] == glob_df['Date'][j]:
            tot_hrs_6mth.append(glob_df['Hours_watched'][j])
```

In [236]:
```
noz_glob_6mth['6mth_tot_hrs'] = tot_hrs_6mth
```

In [237]:
```
noz_glob_6mth['%_hrs_watched_6mth'] = noz_glob_6mth['Hours_watched_6mth']/noz_
```

In [238]:
```
X_train, X_test, y_train, y_test = train_test_split(noz_glob_6mth.drop(columns
                                                                       noz_glob_6mth['%_hrs_watch
```

In [239]:
```
rf_6mth = RandomForestRegressor(random_state = 1701)
```

In [240]:
```
'''The best predictor of hours watched 6 months from debut is simply the hours
this feature'''
```

Out[240]:
```
'The best predictor of hours watched 6 months from debut is simply the hours
watched 3 months from debut. No models outperformed\nthis feature'
```

In [241]:
```
best_score = 0
best_grid = 0
for i in ParameterGrid(random_grid):
    rf_6mth.set_params(**i)
    rf_6mth.fit(X_train,y_train)
    y_pred = rf_6mth.predict(X_test)
    # save if best
    if r2_score(y_test, y_pred) > best_score:
        best_score = r2_score(y_test, y_pred)
        best_grid = i

print ("r2: %0.5f" % best_score)
print ("Grid:", best_grid)
```

```
r2: 0.71814
Grid: {'max_depth': 3, 'min_samples_leaf': 5, 'n_estimators': 100}
```

In [242]:
```
'''Predicting the % of total hours watched on Twitch by game has little to no
found. In my opinion these models can be scrapped. In addition, the regression
baseline predictions 1 month in the future. Though moderate, the improvement m
```

Out[242]:
```
'Predicting the % of total hours watched on Twitch by game has little to no u
se. Only a slight improvement in the r2 value was \nfound. In my opinion thes
e models can be scrapped. In addition, the regression analyses only provide m
arginal improvement of\nbaseline predictions 1 month in the future. Though mo
derate, the improvement might be justified based on business need. '
```