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.

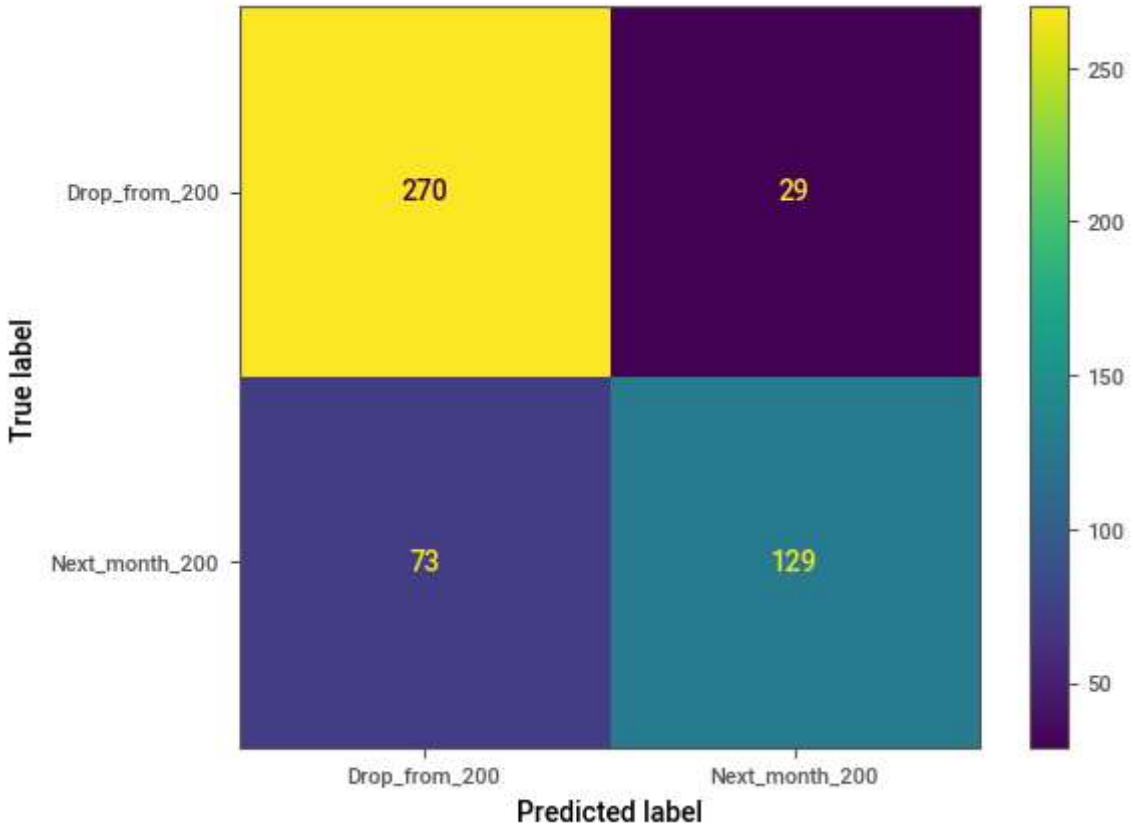
In [20]:
```
print("score on test: " + str(lr.score(X_test_scaled, y_test)))
print("score on train: "+ str(lr.score(X_train_scaled, y_train)))
```

```
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))
```

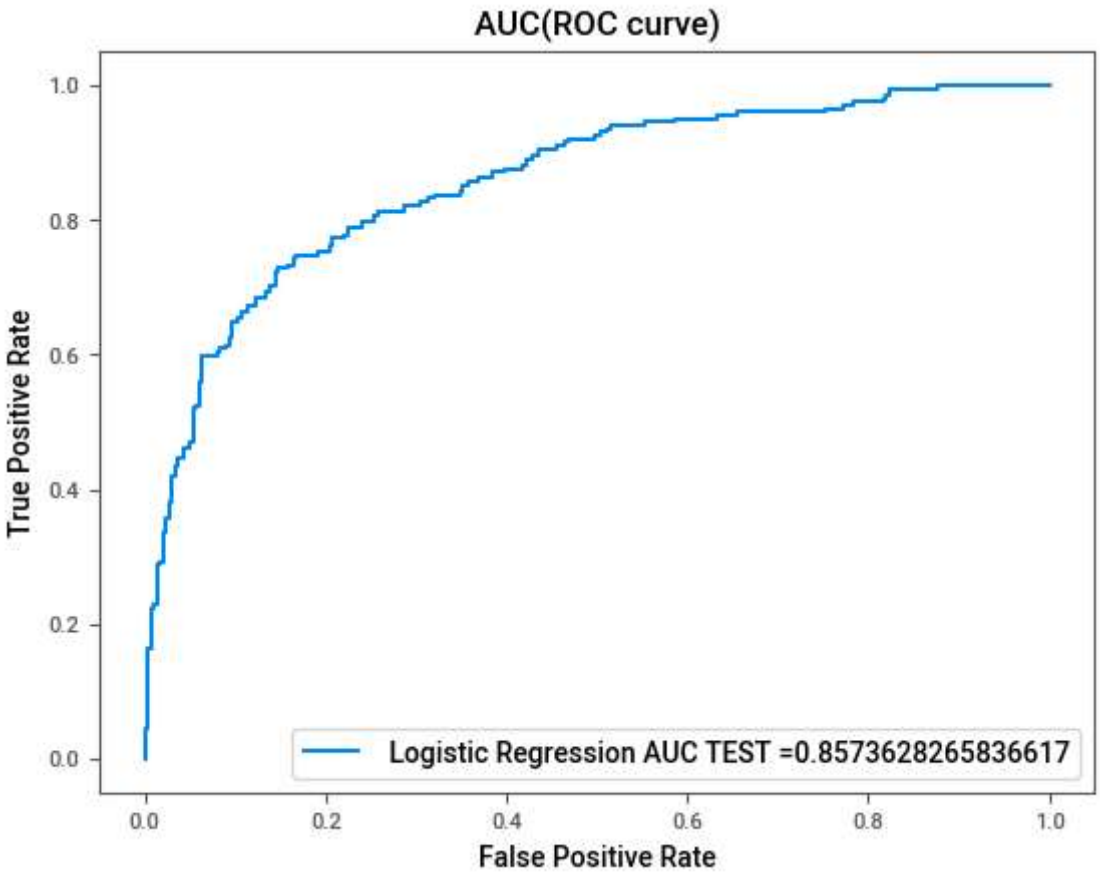
```
              precision    recall  f1-score   support

           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)
```

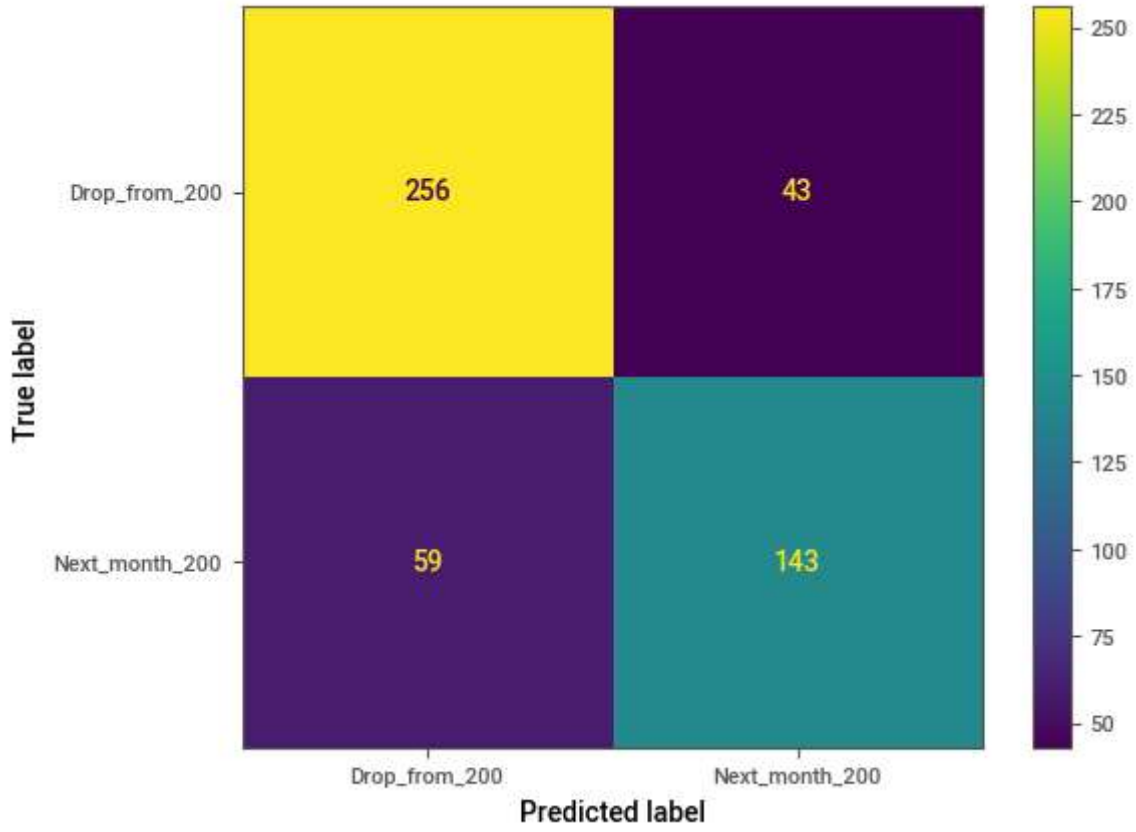
In [27]:
```
plt.plot(fpr_lr, tpr_lr, label=" Logistic Regression AUC TEST ="+str(auc(fpr_l
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC(ROC curve)")
plt.show()
```



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_
```

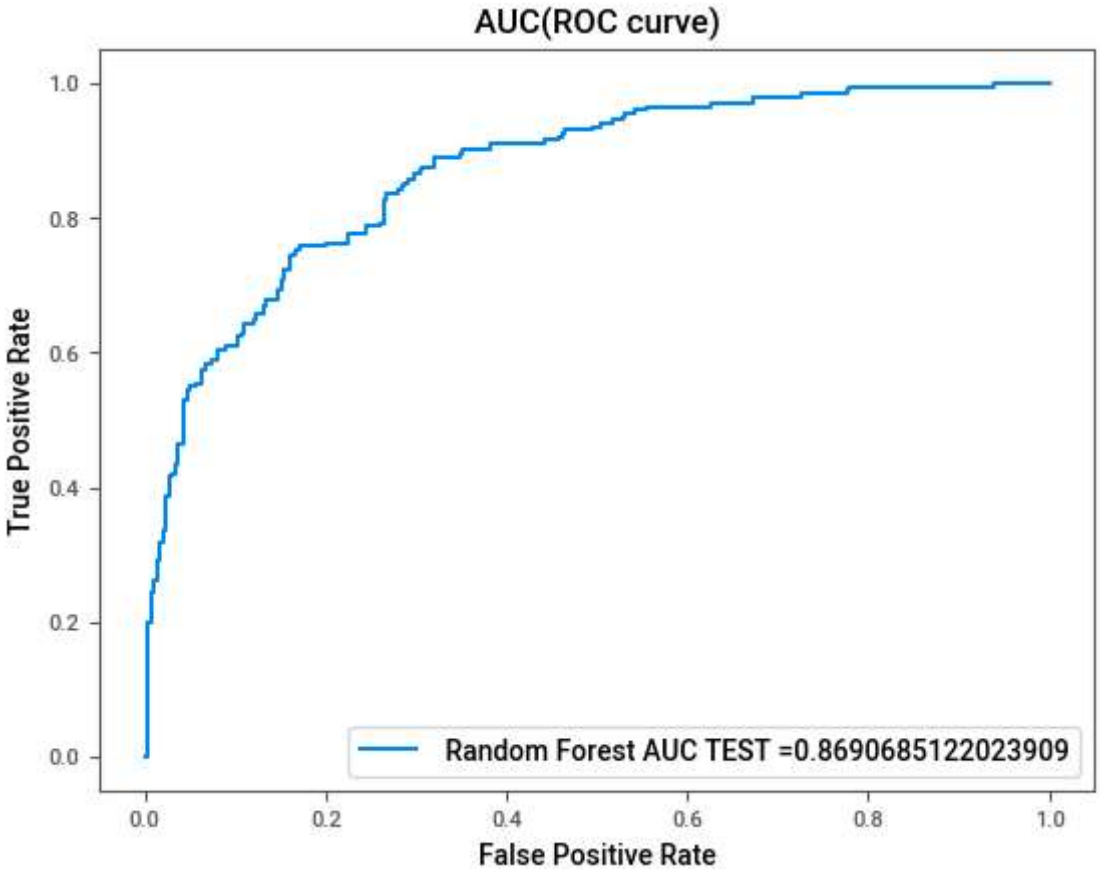
In [47]:
```
rf_importances = pd.Series(importances, index = X_test_scaled.columns)
rf_importances = rf_importances.sort_values(ascending = False)
```

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)
```

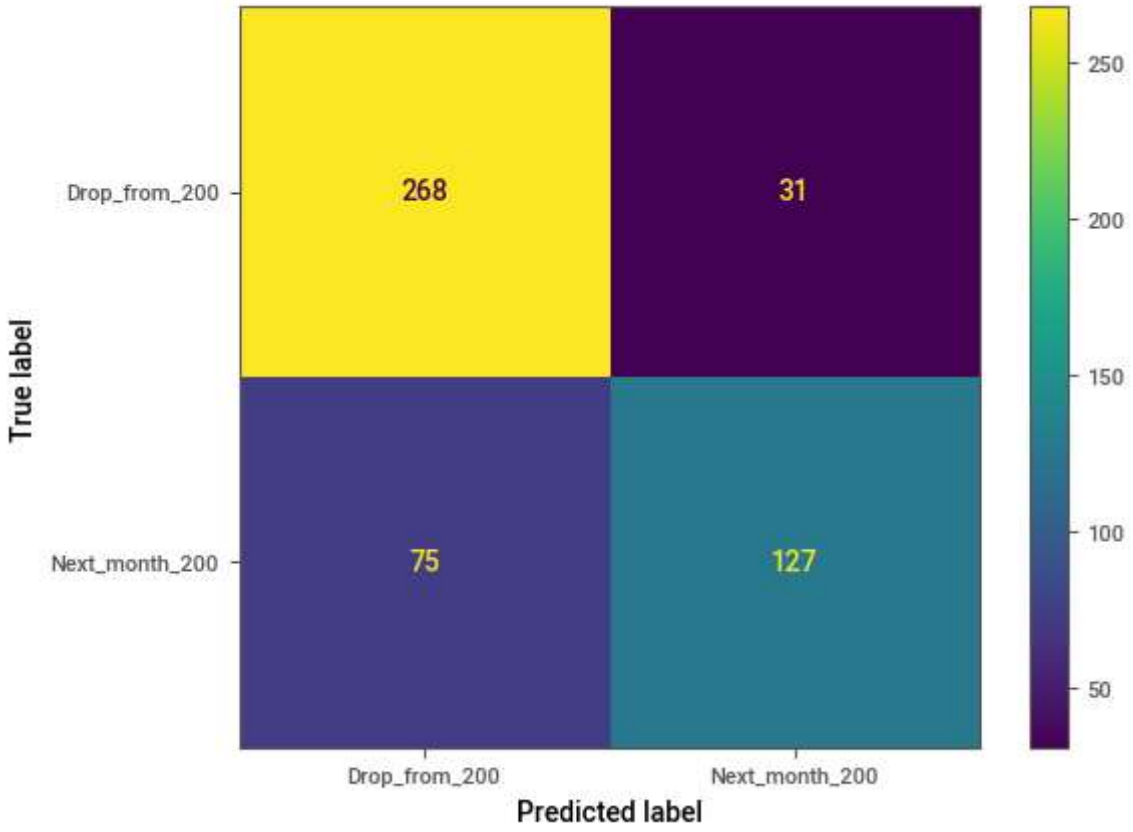
In [50]:
```
plt.plot(fpr_rf, tpr_rf, label=" Random Forest AUC TEST ="+str(auc(fpr_rf, tpr
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC(ROC curve)")
plt.show()
```



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)
```

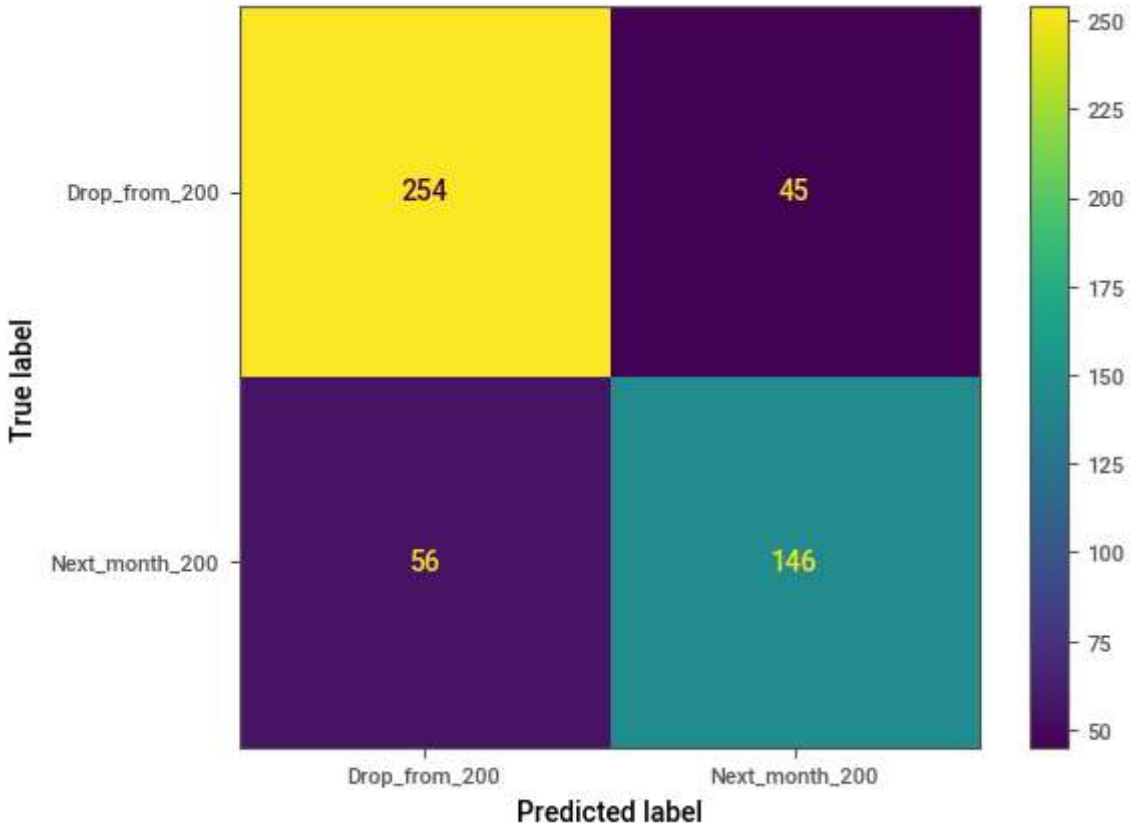
Out[52]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a1f5d17d
00>
```



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. '
```