In [12]:

```
import pandas as pd
import numpy as np
from pandas import DataFrame,Series
import matplotlib.pyplot as plt
import seaborn as sb
import warnings
pd.options.display.max_columns = 1000
data = pd.read_csv('C:\\Users\\niran\\OneDrive\\Desktop\\movie_metadata.csv')
```

In [13]:

```
data.head()
```

Out[13]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_faceb
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	

In [14]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5043 entries, 0 to 5042
Data columns (total 28 columns):
color                      5024 non-null object
director_name              4939 non-null object
num_critic_for_reviews     4993 non-null float64
duration                   5028 non-null float64
director_facebook_likes    4939 non-null float64
actor_3_facebook_likes     5020 non-null float64
actor_2_name               5030 non-null object
actor_1_facebook_likes     5036 non-null float64
gross                      4159 non-null float64
genres                     5043 non-null object
actor_1_name               5036 non-null object
movie_title                5043 non-null object
num_voted_users            5043 non-null int64
cast_total_facebook_likes  5043 non-null int64
actor_3_name               5020 non-null object
facenumber_in_poster       5030 non-null float64
plot_keywords              4890 non-null object
movie_imdb_link            5043 non-null object
num_user_for_reviews       5022 non-null float64
language                   5031 non-null object
country                    5038 non-null object
content_rating             4740 non-null object
budget                     4551 non-null float64
title_year                 4935 non-null float64
actor_2_facebook_likes     5030 non-null float64
imdb_score                 5043 non-null float64
aspect_ratio               4714 non-null float64
```

```
aspect_ratio               4714 non-null float64
movie_facebook_likes       5043 non-null int64
dtypes: float64(13), int64(3), object(12)
memory usage: 1.1+ MB
```

In [15]:

```
data.isnull().sum().sort_values(ascending = False)[:5]
```

Out[15]:

```
gross             884
budget            492
aspect_ratio      329
content_rating    303
plot_keywords     153
dtype: int64
```

In [16]:

```
data.dropna(how = 'any',axis = 0,inplace = True)
```

In [17]:

```
feature_num = data.select_dtypes(exclude=['object']).columns
feature_cat = data.select_dtypes(include=['object']).columns
```

In [18]:

```
data_num = data[feature_num]
data_cat = data[feature_cat]
```

In [19]:

```
data_cat.isnull().sum().sort_values(ascending = False)
```

Out[19]:

```
content_rating     0
country            0
language           0
movie_imdb_link    0
plot_keywords      0
actor_3_name       0
movie_title        0
actor_1_name       0
genres             0
actor_2_name       0
director_name      0
color              0
dtype: int64
```

In [24]:

```
data_num.isnull().sum().sort_values(ascending = False)
```

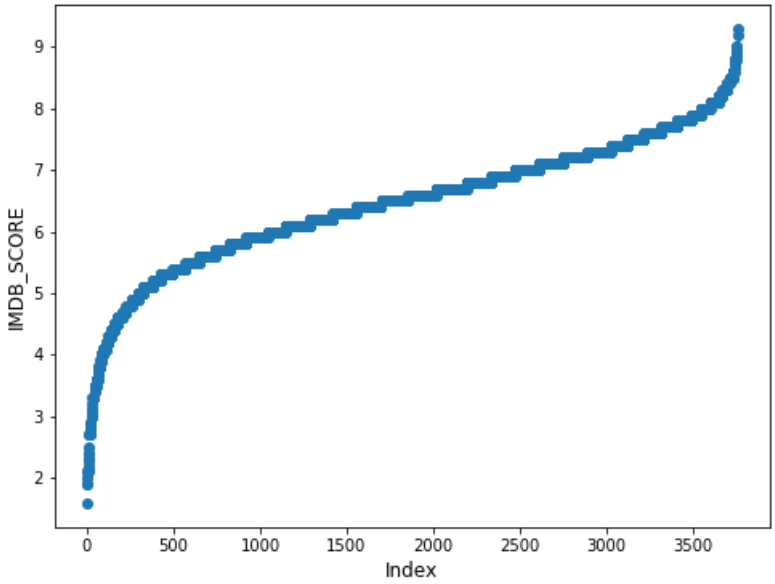
Out[24]:

```
movie_facebook_likes         0
aspect_ratio                 0
imdb_score                   0
actor_2_facebook_likes       0
title_year                   0
budget                       0
num_user_for_reviews         0
facenumber_in_poster         0
cast_total_facebook_likes    0
num_voted_users              0
gross                        0
actor_1_facebook_likes       0
actor_3_facebook_likes       0
```

```
director_facebook_likes      0
duration                     0
num_critic_for_reviews       0
dtype: int64
```

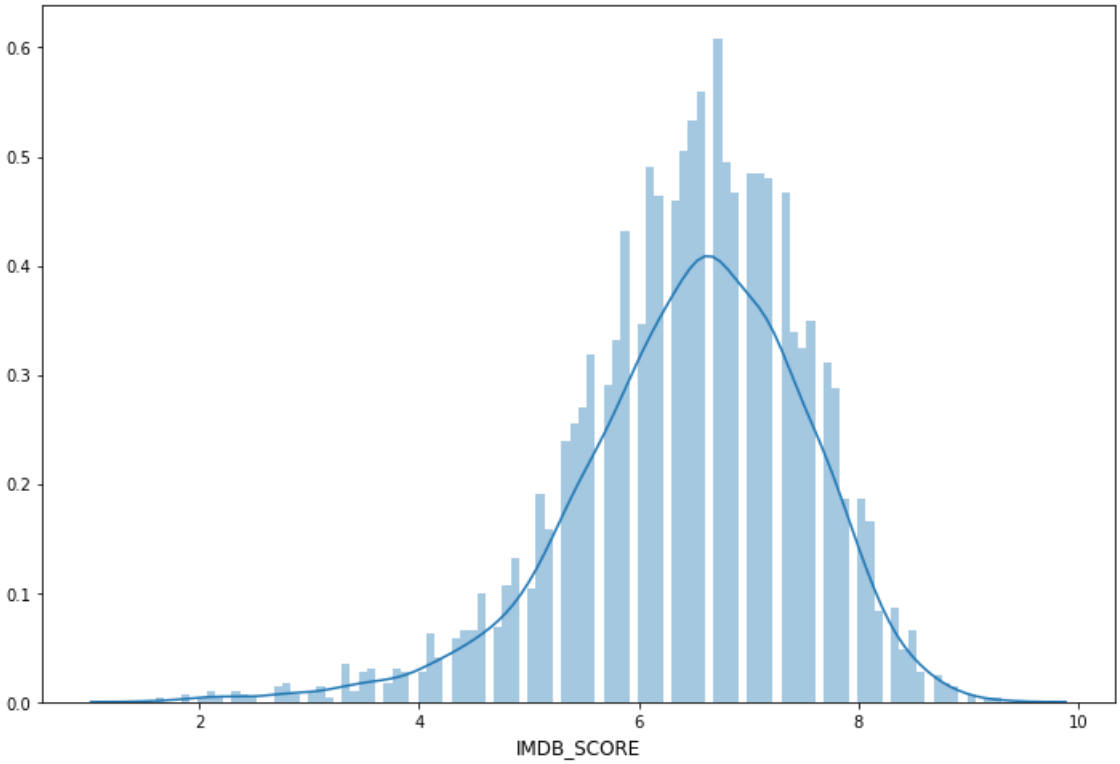
In [25]:

```
plt.figure(figsize=(8,6))
plt.scatter(range(data_num.shape[0]), np.sort(data_num.imdb_score.values))
plt.xlabel('Index', fontsize=12)
plt.ylabel('IMDB_SCORE', fontsize=12)
plt.show()
```



In [26]:

```
plt.figure(figsize=(12,8))
sb.distplot(data_num.imdb_score.values, bins=100)
plt.xlabel('IMDB_SCORE', fontsize=12)
plt.show()
```



Replacing NaN with Median

In [32]:

```
data_num.isnull().sum().sort_values(ascending = False)
data_num.median()
```

Out[32]:

```
num_critic_for_reviews            138.50
duration                          106.00
director_facebook_likes            64.00
actor_3_facebook_likes            436.00
actor_1_facebook_likes           1000.00
gross                          30093107.00
num_voted_users                   53973.50
cast_total_facebook_likes          4059.50
facenumber_in_poster                  1.00
num_user_for_reviews                210.00
budget                         25000000.00
title_year                         2004.00
actor_2_facebook_likes              685.50
imdb_score                            6.60
aspect_ratio                          2.35
movie_facebook_likes                227.00
dtype: float64
```

In [35]:

```
data_num.fillna(data_num.median(),inplace = True)
```

In [39]:

```
correlate = data_num.corr()
high_corr_features = correlate.index[abs(correlate["imdb_score"])>0.1]
```

In [40]:

```
correlate.sort_values(["imdb_score"], ascending = False, inplace = True)
print(correlate.imdb_score)
```

```
imdb_score                 1.000000
num_voted_users            0.482430
duration                   0.366221
num_critic_for_reviews     0.347886
num_user_for_reviews       0.325003
movie_facebook_likes       0.281155
gross                      0.214740
director_facebook_likes    0.192314
cast_total_facebook_likes  0.106803
actor_2_facebook_likes     0.102372
actor_1_facebook_likes     0.093597
actor_3_facebook_likes     0.065544
aspect_ratio               0.029979
budget                     0.029190
facenumber_in_poster      -0.065493
title_year                -0.134982
Name: imdb_score, dtype: float64
```

In [41]:

```
correlate.index[abs(correlate['imdb_score']) > 0.3].tolist()
```

Out[41]:

```
['imdb_score',
 'num_voted_users',
 'duration',
 'num_critic_for_reviews',
 'num_user_for_reviews']
```

Outliers

In [42]:

```
def outliers_detect(val):
    quartile_1, quartile_3 = np.percentile(val, [25, 75])
    iqr = quartile_3 - quartile_1
    lower_bound = quartile_1 - (iqr * 1.5)
    upper_bound = quartile_3 + (iqr * 1.5)
    return np.where((val > upper_bound) | (val < lower_bound))
```

In [45]:

```
test = outliers_detect(data_num['imdb_score'])
test = list(test)

data_num.drop(data_num.index[test],inplace = True)
data_cat.drop(data_cat.index[test],inplace = True)

a = data_num[(data_num.num_voted_users < 10000)].index
data_num.drop(a,inplace = True)
data_cat.drop(a,inplace = True)
```

Categorial Variables & Keyword Plotting

In [46]:

```
df_genres = pd.DataFrame(data_cat['genres'])
df_genres = pd.DataFrame(df_genres.genres.str.split('|').tolist(),columns = ["Genre_"+str(i) for i
in  range(0,8)] )

df_genres = df_genres.reindex(data_cat.index)


data_cat.drop('genres',inplace = True, axis = 1)
data_cat = data_cat.merge(df_genres,left_index = True,right_index = True)
```

In [55]:

```
data_cat.nunique().sort_values()
```

Out[55]:

```
color                 2
Genre_7               3
Genre_6               8
Genre_5              11
content_rating       12
Genre_0              16
Genre_4              16
Genre_3              17
Genre_2              20
Genre_1              21
language             29
country              41
actor_1_name       1172
director_name      1289
actor_2_name       1832
actor_3_name       2215
movie_title        3063
movie_imdb_link    3064
dtype: int64
```

In [57]:

```
data_cat.drop(['movie_imdb_link','Genre_6','Genre_7'],inplace = True, axis = 1)
```

Seperating training and testing data

In [58]:

```
whole_data = pd.concat([data_num,data_num],axis = 1)
y = whole_data['imdb_score']
whole_data.drop('imdb_score',axis = 1,inplace = True)
from sklearn.model_selection import train_test_split # to split the data into two parts
X_train,X_test,y_train,y_test = train_test_split(whole_data,y, random_state = 0,test_size = 0.20) #
test_size = 0.10
num_feat = whole_data.select_dtypes(exclude=['object']).columns.tolist()
cat_feat = whole_data.select_dtypes(include=['object']).columns.tolist()
X_train_num = X_train[num_feat]

X_train_cat = X_train[cat_feat]
X_test_num = X_test[num_feat]

X_test_cat = X_test[cat_feat]
```

Standardization

In [30]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_num_scaled = scaler.fit_transform(X_train_num)
for i, col in enumerate(num_feat):
    X_train_num.loc[:,col] = X_train_num_scaled[:, i]
```

Skewness

In [31]:

```
skewness = X_test_num.apply(lambda x: skew(x.dropna()))
skewness = skewness[abs(skewness) > 0.75]
skew_features = X_test_num[skewness.index]
skew_features  = np.log1p(skew_features)
X_test_num[skewness.index] = skew_features
X_test_num_scaled = scaler.transform(X_test_num)
for i, col in enumerate(num_feat):
    X_test_num.loc[:,col] = X_test_num_scaled[:, i]
```

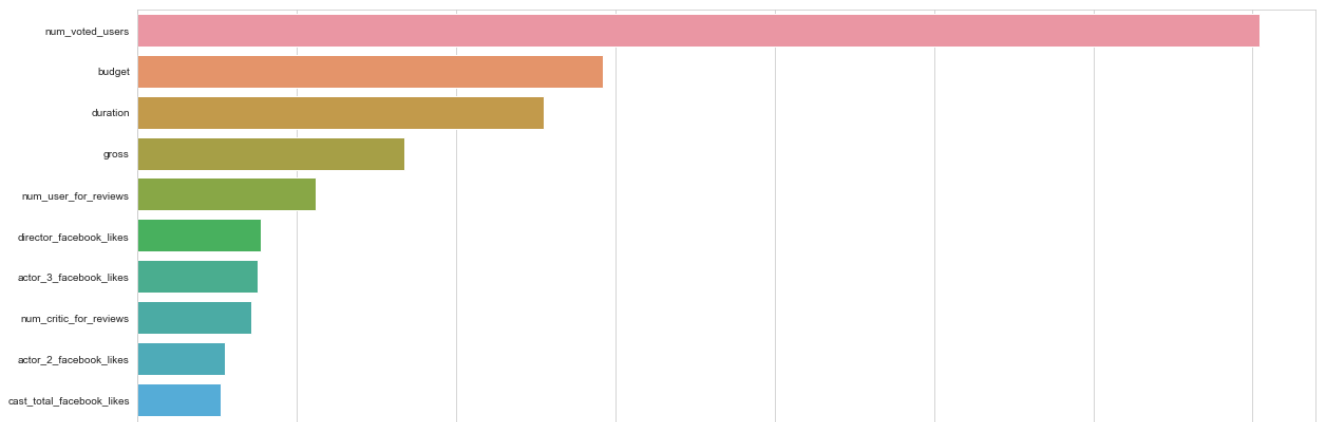
Numeric Feature Importance

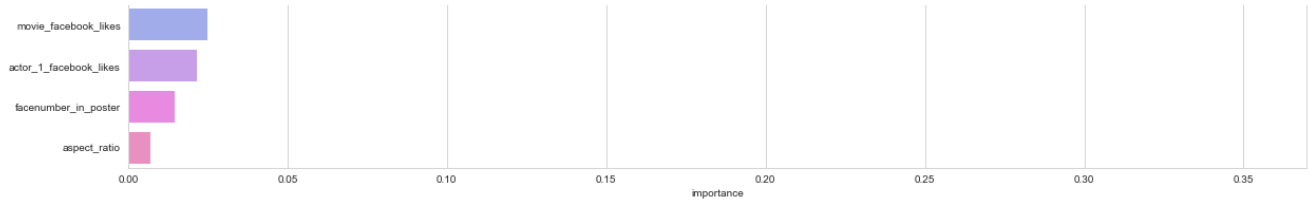
In [34]:

```
df = pd.DataFrame(data = dt.feature_importances_,index = X_train_num.columns.tolist())

df = df[df.iloc[:,0] > 0].sort_values(by = 0,ascending = False)
fig, ax = plt.subplots(figsize=(20,10))
sns.barplot(y = df.index, x= df[0])
plt.xlabel('importance')
```

Out[34]:

```
Text(0.5,0,'importance')
```



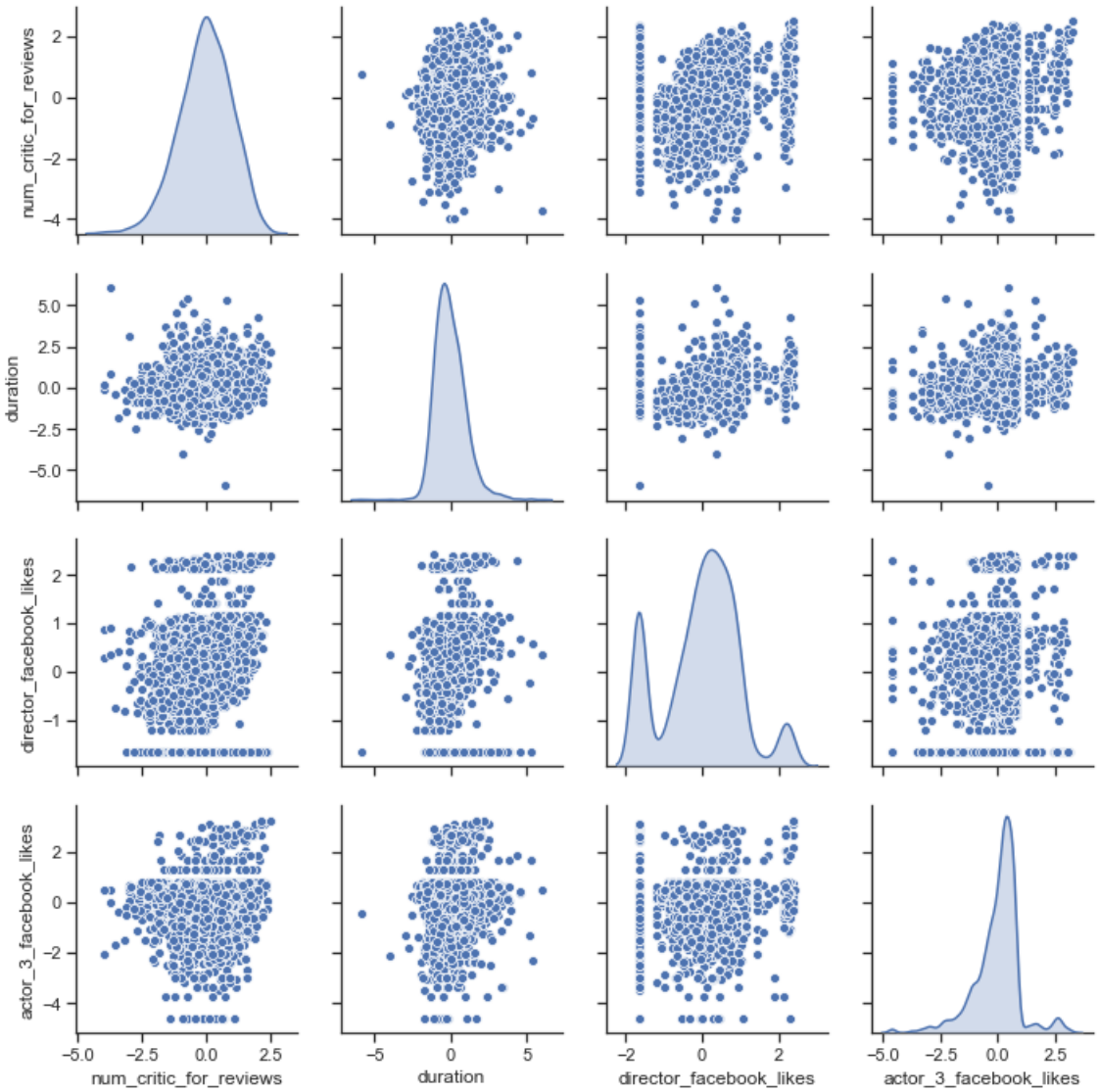


In [35]:

```
sns.set(style="ticks")
sns.pairplot(X_train_num.iloc[:,:4],diag_kind="kde")
```

Out[35]:

```
<seaborn.axisgrid.PairGrid at 0xdf3e358>
```



Encoding

In [36]:

```
train_tar_enc = pd.concat([X_train_cat,y_train],axis = 1)
test_tar_enc = pd.concat([X_test_cat,y_test],axis = 1)
```

Feature Hashing

In [37]:

```
import copy
```

```
X_train_hash = copy.copy(X_train_cat)
X_test_hash = copy.copy(X_test_cat)
from sklearn.feature_extraction import FeatureHasher
for i in range(X_train_cat.shape[1]):
    X_train_hash.iloc[:,i]=X_train_hash.iloc[:,i].astype('str')
for i in range(X_test_hash.shape[1]):
    X_test_hash.iloc[:,i]=X_test_hash.iloc[:,i].astype('str')
h = FeatureHasher(n_features=10000,input_type="string")
```

One Hot Encoding

In [38]:

```
X_train_hash = h.transform(X_train_hash.values)
X_test_hash = h.transform(X_test_hash.values)
```

In [39]:

```
X_train_cat.isnull().sum()
```

Out[39]:

```
color                 0
director_name         0
actor_2_name          0
actor_1_name          0
movie_title           0
actor_3_name          0
language              0
country               0
content_rating        0
Genre_0             498
Genre_1             657
Genre_2            1133
Genre_3            1842
Genre_4            2312
Genre_5            2479
plot_keywords_0     498
plot_keywords_1     499
plot_keywords_2     500
plot_keywords_3     500
plot_keywords_4     502
dtype: int64
```

In [40]:

```
X_train_cat.drop(['Genre_2','Genre_3','Genre_4','Genre_5'],axis = 1,inplace = True)
X_test_cat.drop(['Genre_2','Genre_3','Genre_4','Genre_5'],axis = 1,inplace = True)
```

Lesser Impportant categories coupled as OTHERS

In [41]:

```
temp_cat = pd.concat([X_train_cat,X_test_cat])
```

In [42]:

```
temp_cat.country.value_counts()[:10].index.tolist()
```

Out[42]:

```
['USA',
 'UK',
 'France',
 'Germany',
 'Canada',
 'Australia',
 'Spain',
 'Hong Kong',
 'Japan',
```

```
 'New Zealand']
```

In [43]:

```
temp_cat.loc[temp_cat[~temp_cat["country"].isin(['USA',
 'UK',
 'France',
 'Germany'])].index,"country"] = "Other"
temp_cat.country.value_counts()
```

Out[43]:

```
USA        2528
UK          274
Other       212
France       82
Germany      71
Name: country, dtype: int64
```

In [44]:

```
cat_data.language.value_counts()[:5]
```

Out[44]:

```
English     3047
French        25
Spanish       15
Mandarin      11
German        10
Name: language, dtype: int64
```

In [45]:

```
temp_cat["language"] = (temp_cat["language"] == "English") * 1
temp_cat.language.value_counts()
```

Out[45]:

```
1    3047
0     120
Name: language, dtype: int64
```

In [46]:

```
temp_cat.content_rating.value_counts()[:10]
```

Out[46]:

```
R            1436
PG-13        1143
PG            462
G              64
Not Rated      24
Approved       14
X               9
Unrated         7
NC-17           4
M               2
Name: content_rating, dtype: int64
```

In [47]:

```
temp_cat.loc[temp_cat[(temp_cat["content_rating"] != "R")&(temp_cat["content_rating"] != "PG-13")&(
temp_cat["content_rating"] != "PG")].index,"content_rating"] = "Other"

temp_cat.content_rating.value_counts()
```

Out[47]:

```
R         1436
```

```
PG-13    1143
PG        462
Other     126
Name: content_rating, dtype: int64
```

In [48]:

```
temp_cat.Genre_0.unique()

temp_cat.Genre_0.value_counts()
```

Out[48]:

```
Action         756
Comedy         611
Drama          415
Adventure      276
Crime          168
Biography      132
Horror         106
Animation       36
Fantasy         23
Mystery         14
Documentary      7
Sci-Fi           5
Family           2
Musical          1
Romance          1
Western          1
Name: Genre_0, dtype: int64
```

In [49]:

```
temp_cat.loc[temp_cat[(temp_cat["Genre_0"] != "Action")&(temp_cat["Genre_0"] != "Drama")&(temp_cat[
"Genre_0"] != "Comedy")&(temp_cat["Genre_0"] != "Adventure")&(temp_cat["Genre_0"] != "Crime")&(temp
_cat["Genre_0"] != "Biography")].index,"Genre_0"] = "Other"

temp_cat.Genre_0.value_counts()
```

Out[49]:

```
Other        809
Action       756
Comedy       611
Drama        415
Adventure    276
Crime        168
Biography    132
Name: Genre_0, dtype: int64
```

In [50]:

```
temp_cat.Genre_1.value_counts()

temp_cat.Genre_1.value_counts().index.tolist()

temp_cat.loc[temp_cat[~temp_cat["Genre_1"].isin(['Drama',
 'Adventure',
 'Crime',
 'Comedy',
 'Romance',
 'Mystery',
 'Thriller',
 'Horror',
 'Family',
 'Animation',
 'Fantasy'])].index,"Genre_1"] = "Other"
temp_cat.Genre_1.value_counts()
```

Out[50]:

```
Other        1017
Drama         573
```

```
Adventure     345
Crime         233
Comedy        210
Romance       192
Mystery       125
Fantasy        98
Animation      98
Horror         93
Family         92
Thriller       91
Name: Genre_1, dtype: int64
```

In [51]:

```
temp_cat["color"] = (temp_cat["color"] == "Color") * 1
temp_cat.color.value_counts()

temp_cat.columns.tolist()
```

Out[51]:

```
['color',
 'director_name',
 'actor_2_name',
 'actor_1_name',
 'movie_title',
 'actor_3_name',
 'language',
 'country',
 'content_rating',
 'Genre_0',
 'Genre_1',
 'plot_keywords_0',
 'plot_keywords_1',
 'plot_keywords_2',
 'plot_keywords_3',
 'plot_keywords_4']
```

In [52]:

```
temp_cat.drop(['movie_title'],inplace = True, axis = 1)
```

In [53]:

```
from sklearn.preprocessing import LabelEncoder
abc  = cat_data[[
 'director_name',
 'actor_2_name',
 'actor_1_name',
 'actor_3_name',
 'plot_keywords_0',
 'plot_keywords_1',
 'plot_keywords_2',
'plot_keywords_3',
 'plot_keywords_4']].apply(LabelEncoder().fit_transform)
```

In [55]:

```
temp_cat[[
'director_name',
 'actor_2_name',
 'actor_1_name',
 'actor_3_name',
 'plot_keywords_0',
 'plot_keywords_1',
 'plot_keywords_2','plot_keywords_3',
 'plot_keywords_4']] = abc
```

In [56]:

```
temp_cat = pd.get_dummies(temp_cat)
```

In [57]:

```
X_train_cat = temp_cat.loc[X_train_cat.index,:]

X_test_cat = temp_cat.loc[X_test_cat.index,:]

X_train = pd.concat([X_train_num,X_train_cat], axis =1)

X_test = pd.concat([X_test_num,X_test_cat], axis =1)
```

Random Forest

In [58]:

```
from sklearn.ensemble import RandomForestRegressor
dt = RandomForestRegressor(n_estimators = 1000,n_jobs=-1,random_state = 0)
dt.fit(X_train, y_train)
dt_score_train = dt.score(X_train, y_train)
print("Training score: ",dt_score_train)
dt_score_test = dt.score(X_test, y_test)
print("Testing score: ",dt_score_test)
```

```
('Training score: ', 0.9437708740190368)
('Testing score: ', 0.6040126452004195)
```

Ridge Regression

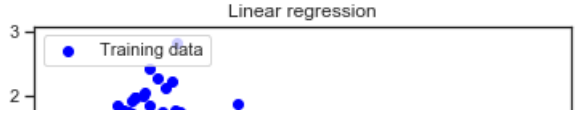
In [59]:

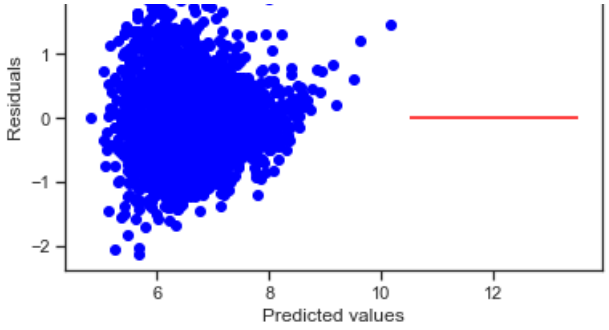
```
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV
ridge = RidgeCV(alphas = [0.01, 0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6, 10, 30, 60])
ridge.fit(X_train,y_train)
alpha = ridge.alpha_
print('best alpha',alpha)
print("Try again for more precision with alphas centered around " + str(alpha))
ridge = RidgeCV(alphas = [alpha * .6, alpha * .65, alpha * .7, alpha * .75, alpha * .8, alpha * .85,

                          alpha * .9, alpha * .95, alpha, alpha * 1.05, alpha * 1.1, alpha * 1.15,
                          alpha * 1.25, alpha * 1.3, alpha * 1.35, alpha * 1.4],cv = 5)
ridge.fit(X_train, y_train)
alpha = ridge.alpha_
print("Best alpha :", alpha)

y_train_rdg = ridge.predict(X_train)
y_test_rdg = ridge.predict(X_test)
print("Training score: ",ridge.score(X_train,y_train))
print("Testing score: ",ridge.score(X_test,y_test))
```

```
('best alpha', 10.0)
Try again for more precision with alphas centered around 10.0
('Best alpha :', 10.5)
('Training score: ', 0.5392456901820885)
('Testing score: ', 0.5119269227555208)
```

In [60]:

```
plt.scatter(y_train_rdg, y_train_rdg - y_train, c = "blue",  label = "Training data")
#plt.scatter(y_test_rdg,y_test_rdg - y_test, c = "green",  label = "Validation data")
plt.title("Linear regression")
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.legend(loc = "upper left")
plt.hlines(y = 0, xmin = 10.5, xmax = 13.5, color = "red")
plt.show()
```



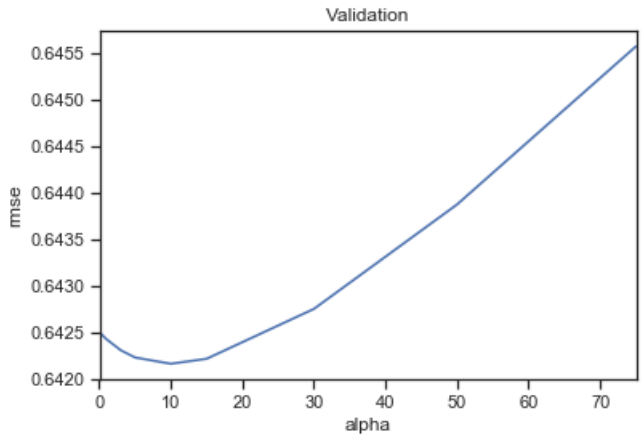


In [62]:

```
from sklearn.model_selection import cross_val_score
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y_train, scoring="neg_mean_squared_error", cv =
5))
    return(rmse)

from sklearn.linear_model import Ridge
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
            for alpha in alphas]

cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

Out[62]:

```
Text(0,0.5,'rmse')
```



In [63]:

```
cv_ridge
```

Out[63]:

```
0.05     0.642496
0.10     0.642492
0.30     0.642476
1.00     0.642422
3.00     0.642303
5.00     0.642226
10.00    0.642158
15.00    0.642212
30.00    0.642748
50.00    0.643870
75.00    0.645569
dtype: float64
```

In [64]:

```
linridge = Ridge(alpha=5).fit(X_train, y_train)
```

In [65]:

```
linridge.score(X_train, y_train)
```

Out[65]:

```
0.5395411334779043
```

Lasso Regression

In [66]:

```
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


#lasso = Lasso(random_state=0)
alphas = np.logspace(-4, -0.5, 30)

tuned_parameters = [{'alpha': alphas}]
n_folds = 3
lasso_cv = LassoCV(alphas=alphas, random_state=0)
# lasso_cv = Lasso(alpha = 0.001)
lasso_cv.fit(X_train, y_train)
#lasso_cv.predict(X_test)
print("Training score: ",lasso_cv.score(X_train, y_train))
print("Testing score: ",lasso_cv.score(X_test, y_test))
```

```
('Training score: ', 0.537443451858346)
('Testing score: ', 0.5103276823901035)
```

In [67]:

```
tuned_parameters = [{'alpha': alphas}]
n_folds = 3
ridge_cv = RidgeCV(alphas=alphas)
ridge_cv.fit(X_train, y_train)
print("Training score: ",ridge_cv.score(X_train, y_train))
print("Testing score: ",ridge_cv.score(X_test, y_test))
```

```
('Training score: ', 0.5396498050801207)
('Testing score: ', 0.5102996859210147)
```

In [68]:

```
temp_whole = pd.concat([X_train,X_test])

temp_whole.shape
target = pd.concat([y_train,y_test])

target_classes = pd.cut(target,bins = [0,6,10],labels = [0,1],right = True,include_lowest = True)

y.size
target_classes.value_counts()
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
pca = PCA(n_components=9, svd_solver="full")
pca_data = pca.fit_transform(temp_whole)
# X_train = pca.transform(X_train)
# pca_data = pca.transform(num_data)
cum_var_exp = np.cumsum(pca.explained_variance_ratio_)

target_classes.isnull().any()
```

Out[68]:

```
False
```

In [69]:

```
from sklearn.model_selection import train_test_split # to split the data into two parts
X_train,X_test,y_train,y_test = train_test_split(temp_whole,target_classes, random_state = 1,test_s
ize = 0.20,stratify =target_classes)
```

In [70]:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()

clf.fit(X_train,y_train)

print('Accuracy of Logistic regression classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.80
Accuracy of Logistic regression classifier on test set: 0.82
```

In [71]:

```
from sklearn.ensemble import RandomForestClassifier
dt = RandomForestClassifier(n_estimators = 1000,n_jobs=-1,random_state = 0)
dt.fit(X_train, y_train)
dt_score_train = dt.score(X_train, y_train)
print("Training score: ",dt_score_train)
dt_score_test = dt.score(X_test, y_test)
print("Testing score: ",dt_score_test)
```

```
('Training score: ', 1.0)
('Testing score: ', 0.8170347003154574)
```

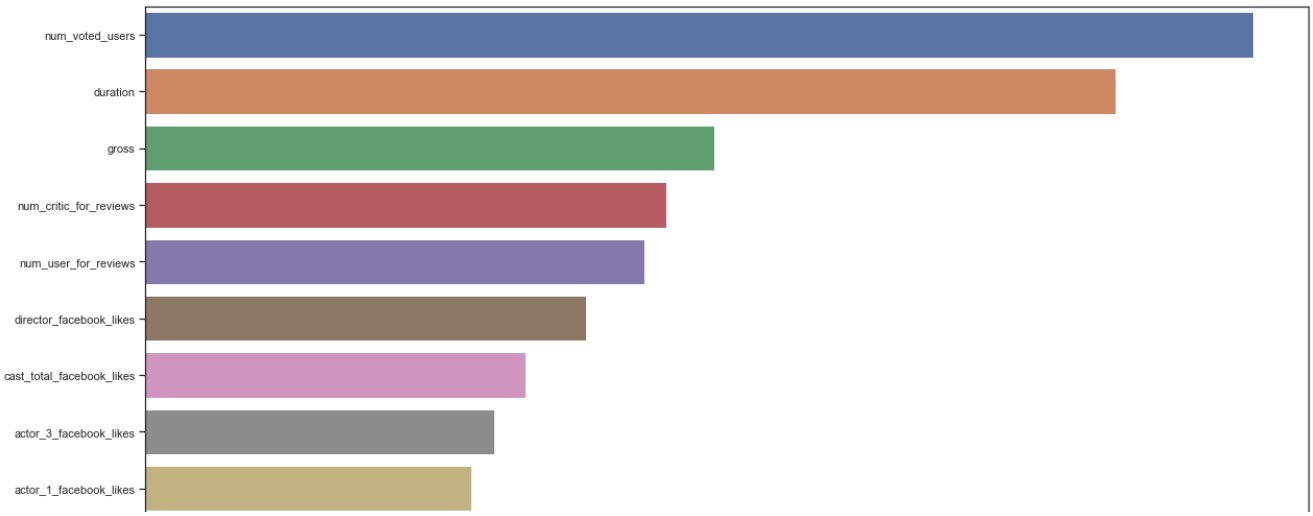
Feature Importance

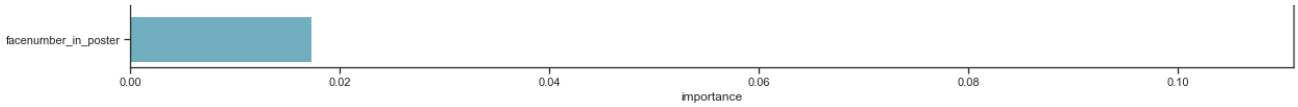
In [72]:

```
df = pd.DataFrame(data = dt.feature_importances_[:10],index = temp_whole.columns.tolist()[:10])

df = df[df.iloc[:,0] > 0].sort_values(by = 0,ascending = False)
fig, ax = plt.subplots(figsize=(20,10))
sns.barplot(y = df.index, x= df[0])
plt.xlabel('importance')
```

Out[72]:

```
Text(0.5,0,'importance')
```





In [90]:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C = 0.1,penalty='l2', random_state=0)

clf.fit(X_train,y_train)

print('Accuracy of Logistic regression classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.80
Accuracy of Logistic regression classifier on test set: 0.82
```

Confusion Matrix

In [91]:

```
from sklearn.metrics import confusion_matrix
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(cm)
```

```
[[ 90  77]
 [ 37 430]]
```