In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from tqdm import tqdm
from datetime import datetime
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [3]:

Out[3]:

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	original_t
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	14000000	[{'id': 35, 'name': 'Comedy'}]	NaN	tt2637294	en	Hot Ti Machin
1	2	[{'id': 107674, 'name': 'The Princess Diaries ...	40000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt0368933	en	The Princ Diarie Rc Engagem
2	3	NaN	3300000	[{'id': 18, 'name': 'Drama'}]	http://sonyclassics.com /whiplash/	tt2582802	en	Whipl
3	4	NaN	1200000	[{'id': 53, 'name': 'Thriller'}, {'id': 18, 'n...	http://kahaanithefilm.com/	tt1821480	hi	Kaha
4	5	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 53, 'nam...	NaN	tt1380152	ko	마린

5 rows × 23 columns

Featurization

In [5]:
```
def prepare(train):
    train[['release_month','release_day','release_year']]=train['release_date'].str.s
    train['release_year'] = train['release_year']

    releaseDate = pd.to_datetime(train['release_date'])
    train['release_dayofweek'] = releaseDate.dt.dayofweek
    train['release_quarter'] = releaseDate.dt.quarter



    train['originalBudget'] = train['budget']
    #Inflation simple formula
    train['inflationBudget'] = train['budget'] + train['budget']*1.8/100*(2018-train[
    train['budget'] = np.log1p(train['budget'])

    #popularity mean year
    train['_popularity_mean_year'] = train['popularity'] / train.groupby("release_yea
    #ratio of Budget to runtime
    train['_budget_runtime_ratio'] = train['budget']/train['runtime']
    #ratio of Budget to popolarity
    train['_budget_popularity_ratio'] = train['budget']/train['popularity']
    #budget year ratio
    train['_budget_year_ratio'] = train['budget']/(train['release_year']*train['relea
    #eleaseYear popularity ratio
    train['_releaseYear_popularity_ratio'] = train['release_year']/train['popularity'
    train['_releaseYear_popularity_ratio2'] = train['popularity']/train['release_year


    #no. of words on title
    train['title_word_count'] = train['title'].str.split().str.len()
    #no. of words on overview
    train['overview_word_count'] = train['overview'].str.split().str.len()
    #no. of words on tagline
    train['tagline_word_count'] = train['tagline'].str.split().str.len()


    #mean of runtime By Year
    train['meanruntimeByYear'] = train.groupby("release_year")["runtime"].aggregate('
    #mean of Popularity By Year
    train['meanPopularityByYear'] = train.groupby("release_year")["popularity"].aggre
    #mean of Budget By Year
    train['meanBudgetByYear'] = train.groupby("release_year")["budget"].aggregate('me
    #median of Budget By Year
    train['medianBudgetByYear'] = train.groupby("release_year")["budget"].aggregate('
    # log budget
    train['log_budget'] = np.log1p(train['budget'])
    return train
```

In [6]:

Out[6]:

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	16.454568	[{'id': 35, 'name': 'Comedy'}]	NaN	tt2637294	
1	2	[{'id': 107674, 'name': 'The Princess Diaries ...	17.504390	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt0368933	
2	3	NaN	15.009433	[{'id': 18, 'name': 'Drama'}]	http://sonyclassics.com/whiplash/	tt2582802	
3	4	NaN	13.997833	[{'id': 53, 'name': 'Thriller'}, {'id':	http://kahaanithefilm.com/	tt1821480	

In [7]:

Out[7]:

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_language	original_
0	1	[{'id': 313576, 'name': 'Hot Tub Time Machine ...	16.454568	[{'id': 35, 'name': 'Comedy'}]	NaN	tt2637294	en	Hot T Machi
1	2	[{'id': 107674, 'name': 'The Princess Diaries ...	17.504390	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	tt0368933	en	The Princ Diarie R Engagen
2	3	NaN	15.009433	[{'id': 18, 'name': 'Drama'}]	http://sonyclassics.com /whiplash/	tt2582802	en	Whipl
3	4	NaN	13.997833	[{'id': 53, 'name': 'Thriller'}, {'id': 18, 'n...	http://kahaanithefilm.com/	tt1821480	hi	Kah
4	5	NaN	0.000000	[{'id': 28, 'name': 'Action'}, {'id': 53, 'nam...	NaN	tt1380152	ko	마린

5 rows × 44 columns

In [9]:

In [10]:

```
train = train.drop(['belongs_to_collection','genres','homepage','imdb_id','overview',
    ,'poster_path','production_companies','production_countries','release_date','spok
    ,'status','title','Keywords','cast','crew','original_language','original_title','
```

In [11]:
```
train.head()
```

Out[11]:

	id	budget	popularity	revenue	release_month	release_day	release_year	release_dayofweek	release_qua
0	1	16.454568	6.575393	12314651	2	20	15	4	
1	2	17.504390	8.248895	95149435	8	6	4	4	
2	3	15.009433	64.299990	13092000	10	10	14	4	
3	4	13.997833	3.174936	16000000	3	9	12	4	
4	5	0.000000	1.148070	3923970	2	5	9	3	

5 rows × 25 columns

In [12]:
```
def prepare(train):
    train[['release_month','release_day','release_year']]=train['release_date'].str.s
    train['release_year'] = train['release_year']

    releaseDate = pd.to_datetime(train['release_date'])
    train['release_dayofweek'] = releaseDate.dt.dayofweek
    train['release_quarter'] = releaseDate.dt.quarter



    train['originalBudget'] = train['budget']
    train['inflationBudget'] = train['budget'] + train['budget']*1.8/100*(2018-train[
    train['budget'] = np.log1p(train['budget'])


    train['_popularity_mean_year'] = train['popularity'] / train.groupby("release_yea
    train['_budget_runtime_ratio'] = train['budget']/train['runtime']
    train['_budget_popularity_ratio'] = train['budget']/train['popularity']
    train['_budget_year_ratio'] = train['budget']/(train['release_year']*train['relea
    train['_releaseYear_popularity_ratio'] = train['release_year']/train['popularity'
    train['_releaseYear_popularity_ratio2'] = train['popularity']/train['release_year





    train['title_word_count'] = train['title'].str.split().str.len()
    train['overview_word_count'] = train['overview'].str.split().str.len()
    train['tagline_word_count'] = train['tagline'].str.split().str.len()



    train['meanruntimeByYear'] = train.groupby("release_year")["runtime"].aggregate('
    train['meanPopularityByYear'] = train.groupby("release_year")["popularity"].aggre
    train['meanBudgetByYear'] = train.groupby("release_year")["budget"].aggregate('me
    train['medianBudgetByYear'] = train.groupby("release_year")["budget"].aggregate('

    train['log_budget'] = np.log1p(train['budget'])
    return train
```

In [13]:

Out[13]:

	id	belongs_to_collection	budget	genres	homepage	imdb_id	original_la
0	3001	[{'id': 34055, 'name': 'Pokémon Collection', '...	0.000000	[{'id': 12, 'name': 'Adventure'}, {'id': 16, '...	http://www.pokemon.com/us/movies/movie-pokemon...	tt1226251	
1	3002	NaN	11.385103	[{'id': 27, 'name': 'Horror'}, {'id': 878, 'na...	NaN	tt0051380	
2	3003	NaN	0.000000	[{'id': 35, 'name': 'Comedy'}, {'id': 10749, '...	NaN	tt0118556	
3	3004	NaN	15.732433	[{'id': 18, 'name': 'Drama'}, {'id': 10752,	http://www.sonyclassics.com/incendies/	tt1255953	

In [14]:

```
test = test.drop(['belongs_to_collection','genres','homepage','imdb_id','overview','r
    ,'poster_path','production_companies','production_countries','release_date','spok
    ,'status','title','Keywords','cast','crew','original_language','original_title','
```

In [15]:

Out[15]:

	id	budget	popularity	release_month	release_day	release_year	release_dayofweek	release_quarter	ori
0	3001	0.000000	3.851534	7	14	7	5.0	3.0	
1	3002	11.385103	3.559789	5	19	58	6.0	2.0	
2	3003	0.000000	8.085194	5	23	97	4.0	2.0	
3	3004	15.732433	8.596012	9	4	10	5.0	3.0	
4	3005	14.508658	3.217680	2	11	5	4.0	1.0	

5 rows × 24 columns

In [19]:

```
from sklearn.model_selection import train_test_split, KFold
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import mean_squared_error

import time
from datetime import datetime
x = train.drop(['id', 'revenue'], axis=1)
y = train['revenue']
X_test = test.drop(['id'], axis=1)
```

In [27]:

```
from sklearn.model_selection import train_test_split, KFold
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import mean_squared_error

import time
from datetime import datetime
X_train = train.drop(['id', 'revenue'], axis=1)
Y_train = train['revenue']
```

In [21]:

```
[illegible]
```

Out[21]:

	budget	popularity	release_month	release_day	release_year	release_dayofweek	release_quarter	origin
1991	17.426428	3.742824	10	31	97	4	4	3
2532	19.336971	23.253089	11	26	12	0	4	25
1074	0.000000	7.631049	4	23	4	4	2	
2696	15.424949	8.679349	3	29	85	4	1	
215	17.370859	7.321172	2	7	13	3	1	3

5 rows × 23 columns

In [22]:

```
[illegible]
```

Out[22]:

```
1991       6482195
2532    1021103568
1074      10000000
2696      27400000
215      173965010
Name: revenue, dtype: int64
```

In [23]:

```
[illegible]
```

Out[23]:

	budget	popularity	release_month	release_day	release_year	release_dayofweek	release_quarter	origina
91	0.00000	3.831960	3	16	1	4	1	
463	0.00000	8.098060	7	31	89	0	3	
2283	0.00000	4.402840	8	8	12	2	3	
2783	17.39903	8.867562	2	15	2	4	1	36
774	0.00000	14.679860	3	12	10	4	1	

5 rows × 23 columns

xgboost

In [30]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats
from scipy.stats import randint as sp_randint
def xg_reg(x_train,x_valid, y_train):
    #Fine tuning
    c_param={'learning_rate' :stats.uniform(0.01,0.2),
      'n_estimators':sp_randint(10,1000),
      'max_depth':sp_randint(1,80),
      'min_child_weight':sp_randint(1,50),
      'gamma':stats.uniform(0,0.04),
      'subsample':stats.uniform(0.6,0.4),
      'reg_alpha':sp_randint(0,200),
      'reg_lambda':stats.uniform(0,200),
      'colsample_bytree':stats.uniform(0.6,0.3)}

    xreg= xgb.XGBRegressor(nthread = 4)
    #RandomSearchCV
    model3 = RandomizedSearchCV(xreg, param_distributions= c_param, scoring = "neg_me

    model3.fit(x_train, y_train,eval_set=[(x_train, y_train), (x_valid, y_valid)], ev
        verbose=1000, early_stopping_rounds=200)

    y_pred = model3.predict(x_valid)
    xgb_test_predictions = [round(value) for value in y_pred]
    y_pred = model3.predict(x_train)
    xgb_train_predictions = [round(value) for value in y_pred]
    print(model3.best_params_)
```

In [31]:

```
[0]     validation_0-rmse:1.426e+08     validation_1-rmse:1.59397e+08
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for early
stopping.

Will train until validation_1-rmse hasn't improved in 200 rounds.
Stopping. Best iteration:
[82]    validation_0-rmse:6.95275e+07   validation_1-rmse:9.24872e+07

[0]     validation_0-rmse:1.4295e+08    validation_1-rmse:1.59841e+08
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for early
stopping.

Will train until validation_1-rmse hasn't improved in 200 rounds.
Stopping. Best iteration:
[276]   validation_0-rmse:5.569e+07     validation_1-rmse:8.96276e+07

[0]     validation_0-rmse:1.42183e+08   validation_1-rmse:1.58755e+08
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for early
stopping.
```

In [32]:
```
xgbmodel = xgb.XGBRegressor(colsample_bytree= 0.7353097200993632, gamma= 0.0029118642
                            learning_rate= 0.17028661557759184, max_depth=56 ,
                            min_child_weight= 20, n_estimators= 16, reg_alpha= 25,
```

In [33]:
```
def rmsle(predictions, dmat):
    labels = dmat.get_label()
    diffs = numpy.log(predictions + 1) - numpy.log(labels + 1)
    squared_diffs = numpy.square(diffs)
    avg = numpy.mean(squared_diffs)
```

In [34]:
```
xgbmodel.fit(x_train, y_train,
        eval_set=[(x_train, y_train), (x_valid, y_valid)], eval_metric='rmse',
```

```
[0]	validation_0-rmse:1.38111e+08	validation_1-rmse:1.54236e+08
Multiple eval metrics have been passed: 'validation_1-rmse' will be used for early
stopping.

Will train until validation_1-rmse hasn't improved in 200 rounds.
[15]	validation_0-rmse:7.75284e+07	validation_1-rmse:9.67881e+07
```

Out[34]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.7353097200993632, gamma=0.002911864213301354,
       importance_type='gain', learning_rate=0.17028661557759184,
       max_delta_step=0, max_depth=56, min_child_weight=20, missing=None,
       n_estimators=16, n_jobs=1, nthread=None, objective='reg:linear',
       random_state=0, reg_alpha=25, reg_lambda=36.04295414564442,
       scale_pos_weight=1, seed=None, silent=True,
       subsample=0.9170549552402483)
```

In [38]:
```
start1=time.time()
valid1=xgbmodel.predict(x_valid)
end1= time.time()
t1=end1-start1
```

Out[38]:
```
0.10264778137207031
```

In [39]:

Out[39]:
```
0.5883303905253772
```

In [40]:

Out[40]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.7353097200993632, gamma=0.002911864213301354,
       importance_type='gain', learning_rate=0.17028661557759184,
       max_delta_step=0, max_depth=56, min_child_weight=20, missing=None,
       n_estimators=16, n_jobs=1, nthread=None, objective='reg:linear',
       random_state=0, reg_alpha=25, reg_lambda=36.04295414564442,
       scale_pos_weight=1, seed=None, silent=True,
       subsample=0.9170549552402483)
```

In [44]:
```
#saving predictions to csv
test['revenue'] = xgbmodel.predict(X_test)
test[['id','revenue']].to_csv('submission_xgb.csv', index=False)
```

Out[44]:

	id	revenue
0	3001	8141865.0
1	3002	5390163.0
2	3003	25629840.0
3	3004	19291872.0
4	3005	3509860.0

lightgbm

In [45]:
```
import lightgbm as lgb
def lgbmodel(x_train,x_valid, y_train):
    #Fine tuning
    c_param={'learning_rate' :stats.uniform(0.01,0.2),
      'n_estimators':sp_randint(100,1000),
      'num_leaves':sp_randint(1,30),
      'boosting_type' : ['gbdt'],
      'bagging_fraction' : stats.uniform(0.1, 0.8),
      'colsample_bytree':stats.uniform(0,0.2),
      'subsample':stats.uniform(0.8,0.4),
      'reg_alpha':sp_randint(0,200),
      'reg_lambda':stats.uniform(0,200)}

    lgbmodel= lgb.LGBMRegressor(nthread = 4)
    model3 = RandomizedSearchCV(lgbmodel, param_distributions= c_param, scoring = "ne

    model3.fit(x_train, y_train,eval_set=[(x_train, y_train), (x_valid, y_valid)], ev
        verbose=1000, early_stopping_rounds=200)

    y_pred = model3.predict(x_valid)
    lgb_test_predictions = [round(value) for value in y_pred]
    l_pred = model3.predict(x_train)
    lgb_train_predictions = [round(value) for value in y_pred]
    print(model3.best_params_)
```

In [46]:
```
[illegible]
```

```
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[220]   valid_0's l2: 4.96611e+15       valid_1's l2: 8.19653e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[337]   valid_0's l2: 4.30935e+15       valid_1's l2: 7.67629e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[230]   valid_0's l2: 4.58996e+15       valid_1's l2: 7.74392e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[61]    valid_0's l2: 4.75764e+15       valid_1's l2: 8.16181e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[68]    valid_0's l2: 4.44109e+15       valid_1's l2: 7.5824e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[44]    valid_0's l2: 4.84564e+15       valid_1's l2: 7.55667e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[109]   valid_0's l2: 5.02386e+15       valid_1's l2: 8.32118e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[138]   valid_0's l2: 4.57476e+15       valid_1's l2: 7.76414e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[111]   valid_0's l2: 4.62531e+15       valid_1's l2: 7.75466e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[585]   valid_0's l2: 5.10948e+15       valid_1's l2: 8.21057e+15
Training until validation scores don't improve for 200 rounds.
Did not meet early stopping. Best iteration is:
[823]   valid_0's l2: 4.67511e+15       valid_1's l2: 7.65264e+15
Training until validation scores don't improve for 200 rounds.
Did not meet early stopping. Best iteration is:
[823]   valid_0's l2: 4.48144e+15       valid_1's l2: 7.63865e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[173]   valid_0's l2: 5.19301e+15       valid_1's l2: 8.54707e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[164]   valid_0's l2: 5.16437e+15       valid_1's l2: 8.08721e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[111]   valid_0's l2: 5.30747e+15       valid_1's l2: 8.14519e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[549]   valid_0's l2: 5.49673e+15       valid_1's l2: 8.55771e+15
Training until validation scores don't improve for 200 rounds.
Did not meet early stopping. Best iteration is:
[783]   valid_0's l2: 5.12398e+15       valid_1's l2: 8.21503e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[464]   valid_0's l2: 5.37325e+15       valid_1's l2: 8.3346e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[282]   valid_0's l2: 4.64162e+15       valid_1's l2: 8.1515e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[333]   valid_0's l2: 4.42432e+15       valid_1's l2: 7.75141e+15
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
```

In [47]:
```
lgbmodel = lgb.LGBMRegressor(bagging_fraction= 0.3652874372414614,
                             boosting_type= 'gbdt',
                             colsample_bytree= 0.014630156676258946,
                             learning_rate= 0.10249020880854486,
                             n_estimators= 823, num_leaves= 10,
                             reg_alpha= 80, reg_lambda= 17.15256293208538,
```

In [48]:

In [49]:
```
lgbmodel.fit(x_train, y_train,
        eval_set=[(x_train, y_train), (x_valid, y_valid)], eval_metric='rmse',
```

```
Training until validation scores don't improve for 200 rounds.
Early stopping, best iteration is:
[188]	training's l2: 5.0332e+15	training's rmse: 7.09451e+07	valid_1's
l2: 7.6446e+15	valid_1's rmse: 8.74334e+07
```

Out[49]:
```
LGBMRegressor(bagging_fraction=0.3652874372414614, boosting_type='gbdt',
       class_weight=None, colsample_bytree=0.014630156676258946,
       importance_type='split', learning_rate=0.10249020880854486,
       max_depth=-1, min_child_samples=20, min_child_weight=0.001,
       min_split_gain=0.0, n_estimators=823, n_jobs=-1, num_leaves=10,
       objective=None, random_state=None, reg_alpha=80,
       reg_lambda=17.15256293208538, silent=True,
       subsample=1.067666689279384, subsample_for_bin=200000,
       subsample_freq=0)
```

In [50]:
```
start1=time.time()
valid1=lgbmodel.predict(x_valid)
end1= time.time()
t1=end1-start1
```

Out[50]:
```
0.01791691780090332
```

In [51]:

Out[51]:
```
0.6640619056295198
```

In [52]:
```
#saving predictions to csv
test['revenue'] = lgbmodel.predict(X_test)
test[['id','revenue']].to_csv('submission_lgb.csv', index=False)
```

Out[52]:

	id	revenue
0	3001	7.539970e+06
1	3002	-4.053703e+06
2	3003	2.220968e+07
3	3004	-1.591284e+07
4	3005	-3.637682e+07

In [68]:
```
x_train[x_train==np.inf]=np.nan
x_train.fillna(x_train.mean(), inplace=True)
y_train[y_train==np.inf]=np.nan
y_train.fillna(y_train.mean(), inplace=True)x_valid[x_valid==np.inf]=np.nan
```

In [95]:
```
def RF_reg(df_train,df_test,train_output):
    #Fine tuning
    n_est = sp_randint(400,600)
    max_dep = sp_randint(10, 20)
    min_split = sp_randint(8, 15)
    start = [False]
    min_leaf = sp_randint(8, 15)
    c_param = {'n_estimators':n_est ,'max_depth': max_dep,'min_samples_split':min_spl
               'min_samples_leaf':min_leaf ,'warm_start':start }

    RF_reg = RandomForestRegressor(max_features='sqrt',oob_score = TRUE, n_jobs=4)

    model2 =  RandomizedSearchCV(RF_reg, param_distributions= c_param, scoring = "neg

    model2.fit(df_train, train_output)

    y_pred = model2.best_estimator_.predict(df_test)
    rndf_test_predictions = [round(value) for value in y_pred]
    y_pred = model2.best_estimator_.predict(df_train)
    rndf_train_predictions = [round(value) for value in y_pred]
    print(model2.best_params_)
```

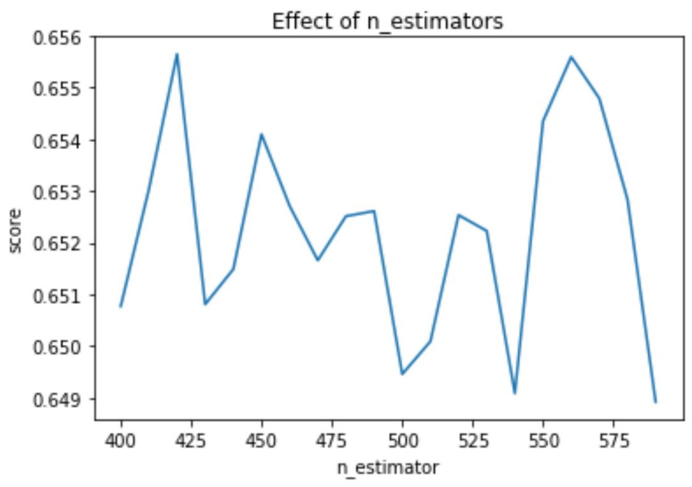
In [71]:
```
from sklearn.ensemble import RandomForestRegressor
```

```
{'max_depth': 18, 'min_samples_leaf': 8, 'min_samples_split': 12, 'n_estimators':
444, 'warm_start': False}
```

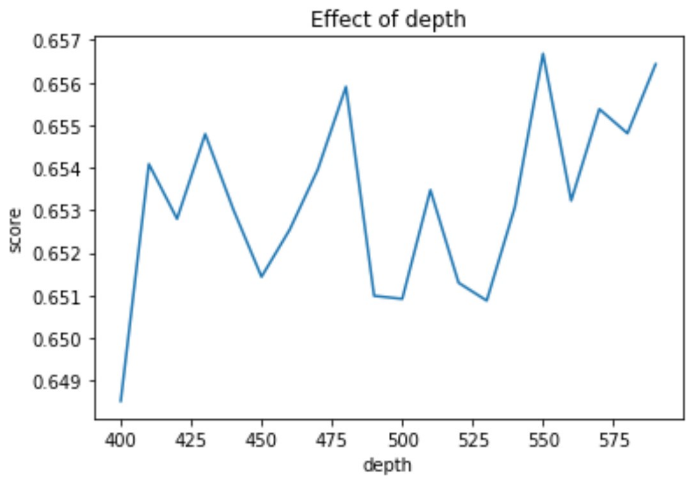
In [80]:
```
X_train[X_train==np.inf]=np.nan
X_train.fillna(X_train.mean(), inplace=True)
Y_train[Y_train==np.inf]=np.nan
Y_train.fillna(Y_train.mean(), inplace=True)
X_test[X_test==np.inf]=np.nan
```

In [92]:
```
estimators = np.arange(400, 600, 10)
scores = []
for n in estimators:
    rfmodel.set_params(n_estimators=n)
    rfmodel.fit(x_train, y_train)
    scores.append(rfmodel.score(x_valid, y_valid))
plt.title("Effect of n_estimators")
plt.xlabel("n_estimator")
plt.ylabel("score")
```

Out[92]:
```
[<matplotlib.lines.Line2D at 0x1e1b8bce470>]
```



In [94]:
```
max_depth = np.arange(1, 20,1)
scores = []
for n in estimators:
    rfmodel.set_params(n_estimators=n)
    rfmodel.fit(x_train, y_train)
    scores.append(rfmodel.score(x_valid, y_valid))
plt.title("Effect of depth")
plt.xlabel("depth")
plt.ylabel("score")
```
Out[94]:
```
[<matplotlib.lines.Line2D at 0x1e1c3f6e668>]
```



In [101]:
```
rfmodel = RandomForestRegressor(n_estimators = 200, min_samples_leaf=8,min_samples_sp
```
In [103]:

Out[103]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=18,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=8, min_samples_split=12,
           min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=4,
           oob_score=True, random_state=None, verbose=0, warm_start=False)
```
In [105]:

In [106]:
```
start1=time.time()
valid1=rfmodel.predict(x_valid)
end1= time.time()
t1=end1-start1
```
Out[106]:
```
0.10871267318725586
```
In [107]:

Out[107]:
```
0.6519483321224786
```
In [77]:

Out[77]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=18,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=8, min_samples_split=12,
           min_weight_fraction_leaf=0.0, n_estimators=444, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [81]:
```
#saving predictions to csv
test['revenue'] = rfmodel.predict(X_test)
test[['id','revenue']].to_csv('submission_rf.csv', index=False)
```

Out[81]:

	id	revenue
0	3001	7.277252e+06
1	3002	1.441477e+07
2	3003	2.869336e+07
3	3004	2.539468e+07
4	3005	9.138782e+06

In [2]:
```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ['S.no', 'Model Name', 'Root Mean Squared Error']
ptable.add_row(["1","Random forest","0.65"])
ptable.add_row(["2","Lightgbm","0.66"])
ptable.add_row(["3","XGBoost","0.58"])
```

```
+------+---------------+-------------------------+
| S.no |   Model Name  | Root Mean Squared Error |
+------+---------------+-------------------------+
|  1   | Random forest |           0.65          |
|  2   |    Lightgbm   |           0.66          |
|  3   |    XGBoost    |           0.58          |
+------+---------------+-------------------------+
```

In [ ]: