In [719…
```
import numpy as np
import pandas as pd

import math
import statistics
import scipy.stats

from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk.corpus import stopwords

from sklearn.feature_selection import f_regression

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import GridSearchCV
```

In [738…
```
data = pd.read_csv("train.csv")
data_test = pd.read_csv("test.csv")
```

In [737…
```
print(data.shape, data_test.shape)
print(data.iloc[0])
```

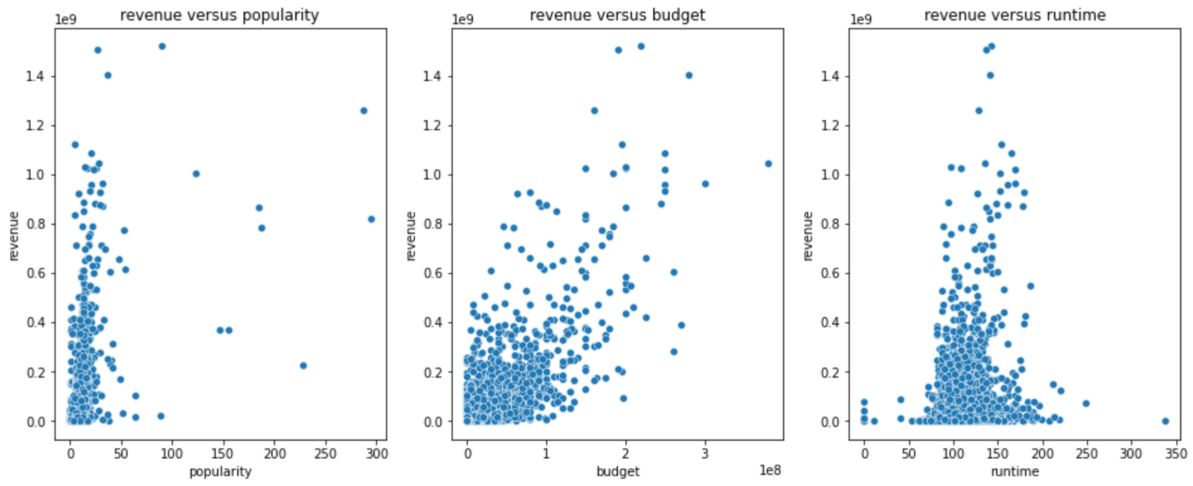
```
(3000, 23) (4398, 22)
id                                                                       1
belongs_to_collection    [{'id': 313576, 'name': 'Hot Tub Time Machine ...
budget                                                            14000000
genres                                      [{'id': 35, 'name': 'Comedy'}]
homepage                                                               NaN
imdb_id                                                          tt2637294
original_language                                                       en
original_title                                         Hot Tub Time Machine 2
overview                 When Lou, who has become the "father of the In...
popularity                                                        6.575393
poster_path                               /tQtWuwvMf0hCc2QR2tkolwl7c3c.jpg
production_companies     [{'name': 'Paramount Pictures', 'id': 4}, {'na...
production_countries     [{'iso_3166_1': 'US', 'name': 'United States o...
release_date                                                       2/20/15
runtime                                                               93.0
spoken_languages                  [{'iso_639_1': 'en', 'name': 'English'}]
status                                                            Released
tagline                  The Laws of Space and Time are About to be Vio...
title                                                  Hot Tub Time Machine 2
Keywords                 [{'id': 4379, 'name': 'time travel'}, {'id': 9...
cast                     [{'cast_id': 4, 'character': 'Lou', 'credit_id...
crew                     [{'credit_id': '59ac067c92514107af02c8c8', 'de...
revenue                                                           12314651
Name: 0, dtype: object
```

In [739…
```
#data cleaning, replace values with NaN for specific columns
## specific columns: release_date, status, runtime
def checkNaN(df,df_test):
    print("Check NaN value for DataFrame:","data_train",'\t', "data_test")
    for i in df.columns[:len(df.columns)-2]:
        print(f"{i:<30}{sum((df[i].isna())):>12}{sum(df_test[i].isna()):>12}
    print()
checkNaN(data,data_test)

mask = data["overview"].isna()
data.loc[mask, "overview"] = ''

mask = data_test["overview"].isna()
data_test.loc[mask, "overview"] = ''

mask = data["tagline"].isna()
data.loc[mask, "tagline"] = ''

mask = data_test["tagline"].isna()
data_test.loc[mask, "tagline"] = ''

mask = data["title"].isna()
data.loc[mask, "title"] = ''

mask = data_test["title"].isna()
data_test.loc[mask, "title"] = ''

mask = data["runtime"].isna()
data.loc[mask, "runtime"] = statistics.mode(data["runtime"])

mask = data_test["release_date"].isna()
data_test.loc[mask, "release_date"] = statistics.mode(data["release_date"])

mask = data_test["runtime"].isna()
data_test.loc[mask, "runtime"] = statistics.mode(data["runtime"])

checkNaN(data,data_test)
```

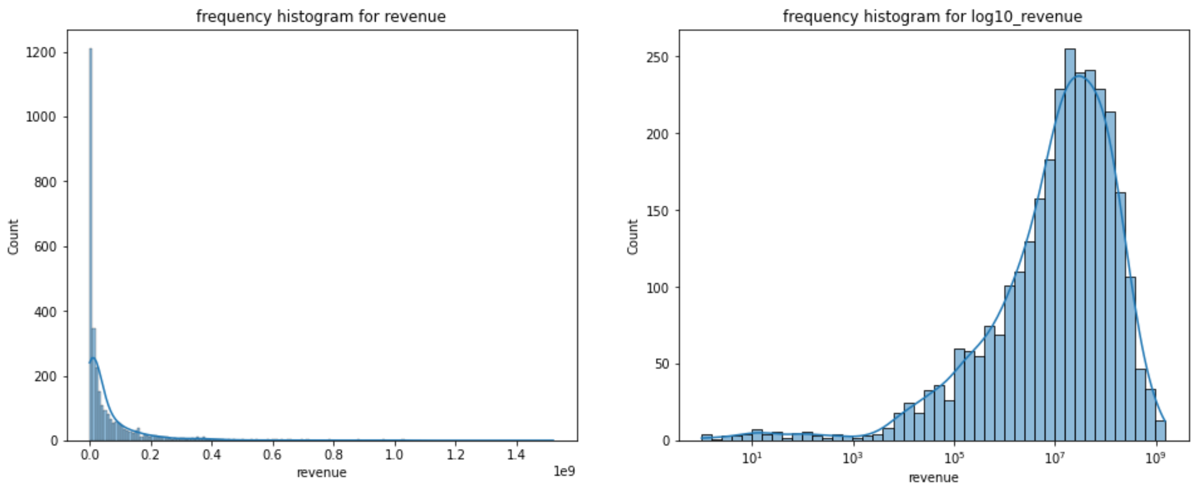
```
Check NaN value for DataFrame: data_train        data_test
id                                      0                0
belongs_to_collection                2396             3521
budget                                  0                0
genres                                  7               16
homepage                             2054             2978
imdb_id                                 0                0
original_language                       0                0
original_title                          0                0
overview                                8               14
popularity                              0                0
poster_path                             1                1
production_companies                  156              258
production_countries                   55              102
release_date                            0                1
runtime                                 2                4
spoken_languages                       20               42
status                                  0                2
tagline                               597              863
title                                   0                3
Keywords                              276              393
cast                                   13               13

Check NaN value for DataFrame: data_train        data_test
id                                      0                0
belongs_to_collection                2396             3521
budget                                  0                0
genres                                  7               16
homepage                             2054             2978
imdb_id                                 0                0
original_language                       0                0
original_title                          0                0
overview                                0                0
popularity                              0                0
poster_path                             1                1
production_companies                  156              258
production_countries                   55              102
release_date                            0                0
runtime                                 0                0
spoken_languages                       20               42
status                                  0                2
tagline                                 0                0
title                                   0                0
Keywords                              276              393
cast                                   13               13
```

In [678…

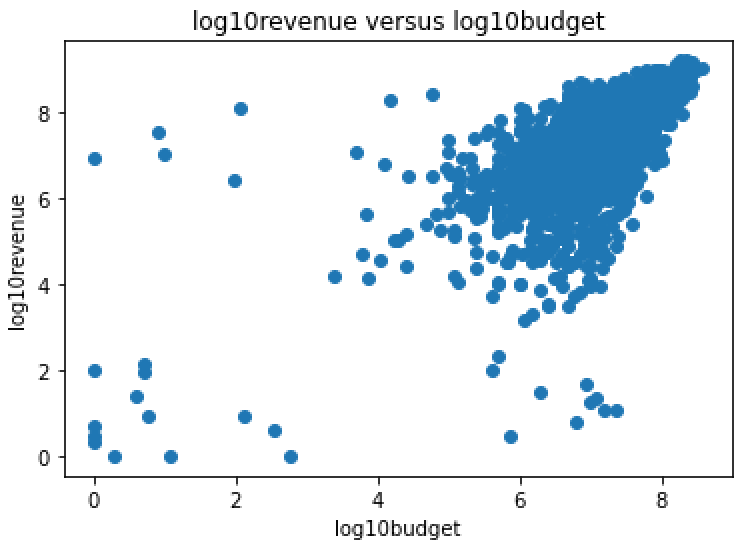
```
#check raw data to determine whether it is reasonable or not
## popularity
## apply boxplot for budget, popularity, revenue
fig, ax = plt.subplots(1,3, figsize = (16,6))

plt.subplot(1,3,1)
sns.scatterplot(x = "popularity", y = "revenue", data = data)
plt.title("revenue versus popularity")
plt.subplot(1,3,2)
sns.scatterplot(x = "budget", y = "revenue", data = data)
plt.title("revenue versus budget")
plt.subplot(1,3,3)
sns.scatterplot(x = "runtime", y = "revenue", data = data)
plt.title("revenue versus runtime")
plt.show()
```



In [715…
```
fig, ax = plt.subplots(1,2, figsize = (16,6))
plt.subplot(1,2,1)
sns.histplot(data = data, x = "revenue", kde = True)
plt.title("frequency histogram for revenue")
plt.subplot(1,2,2)
sns.histplot(data = data, x = "revenue",log_scale = True, kde = True)
plt.title("frequency histogram for log10_revenue")
plt.show()

revenue_log = np.log10(data["revenue"])
```



In [697…
```
def plotScatter(data, y_name, x_name, x_funct = None, y_funct = None):
    x_plt = []
    y_plt = []
    x_label = x_name
    y_label = y_name
    if(x_funct == None):
        x_plt = data[x_name]
    elif (x_funct != None):
        x_plt = x_funct(data[x_name])
        x_label = x_funct.__name__ + x_label

    if(y_funct == None):
        y_plt = data[y_name]
    elif(y_funct != None):
        y_plt = y_funct(data[y_name])
        y_label = y_funct.__name__ + y_label

    plt.scatter(x_plt,y_plt)
    plt.title(y_label + " versus " +x_label)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    return
plotScatter(data, "revenue", "budget",np.log10, np.log10)
```



In [793…

```
def generateTotalList(col, symbol_start, symbol_end):
    res_label_list = []
    res_label = []
    for xi in col:
        res_i = []
        if(type(xi) == str):
            if (symbol_start == None):
                selected = xi
                res_i.append(selected)
                if (selected not in res_label_list):
                    res_label_list.append(selected)

            elif(symbol_start != None):
                count = xi.count(symbol_start)
                for i in range(count):
                    i_start = xi.find(symbol_start)+len(symbol_start)
                    i_end = i_start
                    while(xi[i_end] != symbol_end):
                        i_end += 1
                    selected = xi[i_start:i_end]

                    # add selected label into label list
                    res_i.append(selected)

                    # add new-found label into total label list
                    if(selected not in res_label_list):
                        res_label_list.append(selected)

                    xi = xi[i_end:]
        res_label.append(res_i)
    res_label_list = sorted(res_label_list)
    return res_label, res_label_list

def toYear(col):
    res = []
    for xi in col:
        time_object = datetime.strptime(xi, '%m/%d/%y').date()
        time_year = int(time_object.year)
        if (time_year >= 2023):
            time_year -= 100
        res.append(time_year)
    return res
```

```
def generateBool(col):
    res = []
    for xi in col:
        if(type(xi) == str):
            res.append(1)
        elif(type(xi) != str):
            res.append(0)
    return res

def countItem(col):
    res = []
    for i in col:
        count = len(i)
        res.append(count)
    return res

def generateExpandedEncoder(col,labels):
    res = []
    for xi in col:
        temp_res = []
        for label in labels:
            if label in xi:
                temp_res.append(1)
            elif label not in xi:
                temp_res.append(0)
        res.append(temp_res)
    res = pd.DataFrame(res, columns = labels)
    return res

def fRegressionTest_encoded(encoded_list,name):
    f, p = f_regression(X = encoded_list, y = data['revenue'])
    index = 1
    res_bool = (p < 0.05*index)
    print("--",name,"--")
# #     decrease the p value to further reduce the number of features going
#     while(sum(res_bool) >= 30):
#         print("original No. features:","\t",len(p),"\tfeatures selected:",
#         index *= 0.8
#         res_bool = p < 0.05*index

    print("original No. features:","\t",len(p),"\tfeatures selected:", sum(r
    res = encoded_list[encoded_list.columns[res_bool]]
    return res

def countKeyItem(df):
    res = []
    for i in range(df.shape[0]):
        count_i = sum(df.iloc[i])
        res.append(count_i)
    return res

def generateFDataFrame(list_encoded,selection_encoded,mode = "train",list_f

    list_f_encoded = []
    list_f_columns = []
    list_f_count = []

    if(mode == "train"):
        for encoded_i,name_i in zip(list_encoded,name):
            temp = fRegressionTest_encoded(encoded_i,name_i)
            list_f_encoded.append(temp)
            list_f_columns.append(temp.columns)

    elif(mode == "test"):
        for i in range(len(list_encoded)):
            temp = list_encoded[i].loc[:,list_f[i]]
            list_f_encoded.append(temp)

    list_f_encoded_select = []
    for i in selection_encoded:
        list_f_encoded_select.append(list_f_encoded[i])

    df_f = pd.concat(list_f_encoded_select,axis = 1,join = 'inner')

    return df_f, list_f_columns
```

In [571…

```
# for training data
# extract clean data from string in the form of dictionary
coll_label, coll_list = generateTotalList(data["belongs_to_collection"], "{'
genres_label, genres_list = generateTotalList(data["genres"],"'name': '","'"
prodComp_label, prodComp_list = generateTotalList(data["production_companies
prodCtry_label, prodCtry_list = generateTotalList(data["production_countries
original_label, original_list = generateTotalList(data["original_language"],
spoken_label, spoken_list  = generateTotalList(data["spoken_languages"],"'is
keyword_label, keyword_list = generateTotalList(data["Keywords"],"{'id': ","
cast_label, cast_list = generateTotalList(data["cast"],", 'id': ",",")
crew_label, crew_list = generateTotalList(data["crew"],", 'id': ",",")

# extract release_year from the release_date
release_year = toYear(data["release_date"])

# boolean label :0 or 1
hmpage_label = generateBool(data["homepage"])
poster_label = generateBool(data["poster_path"])

# int function to count the number of the items containned
genres_count = countItem(genres_label)
prodComp_count = countItem(prodComp_label)
prodCtry_count = countItem(prodCtry_label)
spoken_count = countItem(spoken_label)
keyword_count = countItem(keyword_label)
cast_count = countItem(cast_label)
crew_count = countItem(crew_label)

# expand the encoded matrix for features
coll_encoded = generateExpandedEncoder(coll_label, coll_list)
genres_encoded = generateExpandedEncoder(genres_label, genres_list)
prodComp_encoded = generateExpandedEncoder(prodComp_label, prodComp_list)
prodCtry_encoded = generateExpandedEncoder(prodCtry_label, prodCtry_list)
original_encoded = generateExpandedEncoder(original_label, original_list)
spoken_encoded = generateExpandedEncoder(spoken_label, spoken_list)
keyword_encoded = generateExpandedEncoder(keyword_label, keyword_list)
cast_encoded = generateExpandedEncoder(cast_label, cast_list)
crew_encoded = generateExpandedEncoder(crew_label, crew_list)
```

In [764…

```
from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk.corpus import stopwords
stop_words = list(stopwords.words('english'))

overview_cv = CountVectorizer(stop_words = stop_words)
tagline_cv = CountVectorizer(stop_words = stop_words)
title_cv = CountVectorizer(stop_words = stop_words)

#train
overview_vectorized = overview_cv.fit_transform(data["overview"])
tagline_vectorized = tagline_cv.fit_transform(data["tagline"])
```

```
title_vectorized = title_cv.fit_transform(data["title"])

df_overview = pd.DataFrame(overview_vectorized.toarray(),columns = overview_
df_tagline = pd.DataFrame(tagline_vectorized.toarray(), columns = tagline_cv
df_title = pd.DataFrame(title_vectorized.toarray(), columns = title_cv.get_f

#test
overview_vectorized_test = overview_cv.transform(data_test["overview"])
tagline_vectorized_test = tagline_cv.transform(data_test["tagline"])
title_vectorized_test = title_cv.transform(data_test["title"])

df_overview_test = pd.DataFrame(overview_vectorized_test.toarray(),columns =
df_tagline_test = pd.DataFrame(tagline_vectorized_test.toarray(), columns =
df_title_test = pd.DataFrame(title_vectorized_test.toarray(), columns = titl
```

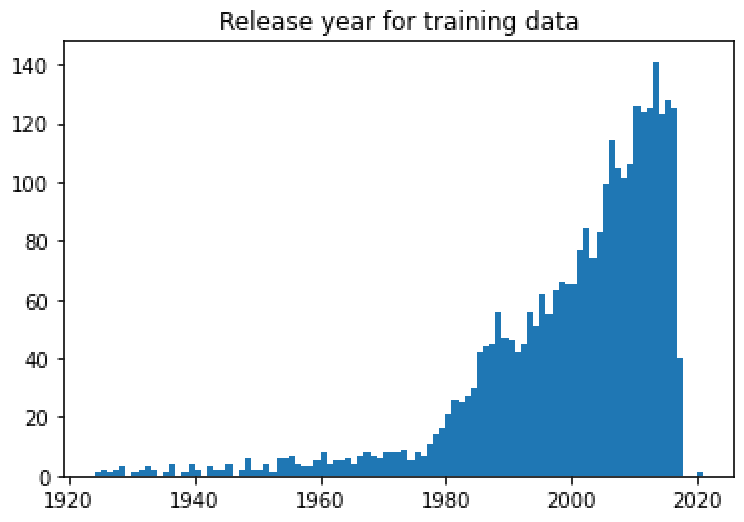
In [572…

```
#for test data
coll_label_test, dummy = generateTotalList(data_test["belongs_to_collection"
genres_label_test, dummy = generateTotalList(data_test["genres"],"'id': ",",
prodComp_label_test, dummy = generateTotalList(data_test["production_compani
prodCtry_label_test, dummy = generateTotalList(data_test["production_countri
original_label_test, dummy = generateTotalList(data_test["original_language"
spoken_label_test, dummy  = generateTotalList(data_test["spoken_languages"],
keyword_label_test, dummy = generateTotalList(data_test["Keywords"],"{'id':
cast_label_test, dummy = generateTotalList(data_test["cast"],", 'id': ",",")
crew_label_test, dummy = generateTotalList(data_test["crew"],", 'id': ",",")

# extract release_year from the release_date
release_year_test = toYear(data_test["release_date"])

# boolean function :0 or 1
hmpage_label_test = generateBool(data_test["homepage"])
overview_label_test = generateBool(data_test["overview"])
poster_label_test = generateBool(data_test["poster_path"])
tagline_label_test = generateBool(data_test["tagline"])

# int function to count the number of the items containned
genres_count_test = countItem(genres_label_test)
prodComp_count_test = countItem(prodComp_label_test)
prodCtry_count_test = countItem(prodCtry_label_test)
spoken_count_test = countItem(spoken_label_test)
keyword_count_test = countItem(keyword_label_test)
cast_count_test = countItem(cast_label_test)
crew_count_test = countItem(crew_label_test)

# expand the encoded matrix for features
coll_encoded_test = generateExpandedEncoder(coll_label_test, coll_list)
genres_encoded_test = generateExpandedEncoder(genres_label_test, genres_list
prodComp_encoded_test = generateExpandedEncoder(prodComp_label_test, prodCom
prodCtry_encoded_test = generateExpandedEncoder(prodCtry_label_test, prodCtr
original_encoded_test = generateExpandedEncoder(original_label_test, origina
spoken_encoded_test = generateExpandedEncoder(spoken_label_test, spoken_list
keyword_encoded_test = generateExpandedEncoder(keyword_label_test, keyword_l
cast_encoded_test = generateExpandedEncoder(cast_label_test, cast_list)
crew_encoded_test = generateExpandedEncoder(crew_label_test, crew_list)
```

In [635…

```
plt.hist(release_year, bins=np.arange(min(release_year), max(release_year)+1
plt.title("Release year for training data")
plt.show()

mmScaler = MinMaxScaler()
df_release_norm = pd.DataFrame(mmScaler.fit_transform(pd.DataFrame(release_y
                               columns = ['release_year_norm'])
```

```
df_release_norm_test = pd.DataFrame(mmScaler.transform(pd.DataFrame(release_
                                    columns = ['release_year_norm'])
```



In [817…

```
#preprocessing for float data

num_scaler = StandardScaler()

data_num_std = [data['budget'],data['popularity']]
df_num_std = pd.DataFrame(data_num_std).T

df_num_std = num_scaler.fit_transform(df_num_std)
df_num_std = pd.DataFrame(df_num_std)
df_num_std.columns = ["budget_std","popularity_std"]

data_num_std_test = [data_test['budget'],data_test['popularity']]
df_num_std_test = pd.DataFrame(data_num_std_test).T

df_num_std_test = num_scaler.fit_transform(df_num_std_test)
df_num_std_test = pd.DataFrame(df_num_std_test)
df_num_std_test.columns = ["budget_std","popularity_std"]
```

In [824…

```
# part of selecting features with p < 0.05 using f_regression
# index
##0-collection          ,f_count
##1-genres              ,f_encoded
##2-prodComp            ,f_count
##3-prodCtry            ,f_encoded
##4-original            ,f_encoded
##5-spoken_languages    ,f_encoded
##6-keywords            ,f_count
##7-cast                ,f_count
##8-crew                ,f_count
##9-overview
##10-tagline
##11_title

list_encoded_train = [coll_encoded, genres_encoded, prodComp_encoded, prodCt
                      original_encoded, spoken_encoded, keyword_encoded, cast_enco

list_encoded_test = [coll_encoded_test, genres_encoded_test, prodComp_encode
                      original_encoded_test, spoken_encoded_test, keyword_encoded_

list_name = ["collection", "genres", "production companies", "production cou
df_f_train, list_f_train = generateFDataFrame(list_encoded_train,
                                              selection_encoded = [0,1,3,4,5
                                              mode = "train",
                                                    list_f = None,
                                                    name = list_name)
df_f_test , dummy = generateFDataFrame(list_encoded_test,
                                                    selection_encoded = [0,1,3,4,5],
                                                    mode = "test",
                                                    list_f = list_f_train,
                                                    name = list_name)
```

```
-- collection --
original No. features:   422    features selected: 28
-- genres --
original No. features:   20     features selected: 12
-- production companies --
original No. features:   3712   features selected: 285
-- production countries --
original No. features:   74     features selected: 16
-- original language --
original No. features:   36     features selected: 12
-- spoken languages --
original No. features:   79     features selected: 8
-- keywords --
original No. features:   7400   features selected: 325
-- cast --
original No. features:   38760  features selected: 1529
-- crew --
original No. features:   38897  features selected: 2336
-- overview --
original No. features:   17301  features selected: 1028
-- tagline --
original No. features:   3184   features selected: 114
-- title --
original No. features:   3297   features selected: 183
```

In [826…

```
# combine all completed proprocessing dataframe to form a dataframe for the
def generateDF_T(lists,colname):
    df = pd.DataFrame(lists).T
    df.columns = colname
    return df

data_bool_label = [hmpage_label, poster_label]
data_bool_label_test = [hmpage_label_test, poster_label_test]
colname_bool = ["homeage","poster"]

df_bool = generateDF_T(data_bool_label, colname_bool)
df_bool_test = generateDF_T(data_bool_label_test, colname_bool)

data_count = [prodComp_count, spoken_count,cast_count,crew_count]
data_count_test = [prodComp_count_test, spoken_count_test,cast_count_test,cr
colname_count = ["prodComp_count","spoken_count", "cast_count", "crew_count"

df_count = generateDF_T(data_count, colname_count)
df_count_test = generateDF_T(data_count_test, colname_count)

df_total = [df_f_train, df_release_norm, df_num_log,df_bool,df_count]
df_total_test = [df_f_test, df_release_norm_test, df_num_log_test, df_bool_t

df_train = pd.concat(df_total,axis = 1, join = "inner")
df_test = pd.concat(df_total_test,axis = 1, join = "inner")
print(df_train.shape,df_test.shape)
```

```
(3000, 85) (4398, 85)
```

In [834…

```
X_train, X_valid, y_train, y_valid = train_test_split(df_train, data["revenu
                                                       test_size = 0.2, randc
```

```
regr = RandomForestRegressor()
param = {'n_estimators': [50,100,150,200,250,500],
         'criterion': ['squared_error', 'friedman_mse'],
         'max_depth': [10,25,50,None]}

gs = GridSearchCV(regr, param, cv = 5, n_jobs = -1,
                  scoring = 'neg_root_mean_squared_error',
                  verbose = 1).fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

In [835…

```
gs.best_params_
```

Out[835]:

```
{'criterion': 'squared_error', 'max_depth': 10, 'n_estimators': 200}
```

In [836…

```
slr = LinearRegression().fit(X_train,y_train)
```

In [837…

```
y_valid_pred_regr = gs.predict(X_valid)
y_valid_pred_slr = slr.predict(X_valid)

def anovaFtest(y, y_pred,n_r, name_model):
    mean_y = statistics.mean(y)
    mean_y_pred = statistics.mean(y_pred)
    ssr = 0
    sse = 0
    n_t = len(y)-1
    n_e = n_t - n_r
    for yi, yi_pred in zip(y,y_pred):
        ssr += (yi_pred - mean_y)**2
        sse += (yi - yi_pred)**2
    msr = ssr/n_r
    mse = sse/n_e
    f_ratio = msr/mse
    f_critical = scipy.stats.f.ppf(q = 1- 0.05, dfn = n_r, dfd = n_t)
    print("---- ANOVA for",name_model,"----")
    print("MSR:",msr,'\t',"MSE:",mse)
    print("F-critical:", f_critical, '\t',"degree of Freedom:", n_r, n_t)
    print("F-value:",f_ratio)

anovaFtest(y_valid,y_valid_pred_regr, X_train.shape[1], "Random Forest Regre
anovaFtest(y_valid,y_valid_pred_slr, X_train.shape[1], "Linear Regression")
```

```
---- ANOVA for Random Forest Regressor ----
MSR: 1.0939018004605838e+17     MSE: 6006424089023688.0
F-critical: 1.2903958928993684   degree of Freedom: 85 599
F-value: 18.212197211642305
---- ANOVA for Linear Regression ----
MSR: 8.242710374058611e+16      MSE: 1.0325165772340464e+16
F-critical: 1.2903958928993684   degree of Freedom: 85 599
F-value: 7.983126427025093
```

In [839…

```
df_y_sub = pd.DataFrame(gs.predict(df_test))
df_id_sub = pd.DataFrame(data_test.iloc[:,0])
submission = pd.concat([df_id_sub,df_y_sub],axis = 1)
submission.columns = ['id', 'revenue']
submission.to_csv("submission_3035790941_4.csv", index = False)
```