Movie User Score & World Gross Analysis

By: Josh Balingit, Luc Chen, Hengyuan (David) Liu, Kathy Mo, Ka Wai Sit

Table of Contents

Introduction & Motivation

Film has become a major form of art, inspiring different writers and directors to tell all sorts of riveting stories. At the same time, films can also be regarded as a form of investment that is extremely dependent on capital and industrial standards. Film has also transformed into a major form of entertainment, consuming countless hours in many people's lives. With how influential films are to creators, investors, and casual movie-goers, it is important to ask what makes a "successful" are to creators, investors, and casual movie-goers, it is important to ask what makes a "successful" film? Naturally, a question like this can be difficult to answer directly since "successful" can be defined in many different ways. In order to address this issue, we decided, for our analysis, to specifically associate a movie's "success" with its user score and world gross. With "success" explicitly defined, we were able to narrow our focus and ask the following questions.

Research Questions

What are the significant factors that impact whether or not a movie gets a good user score?

What are the significant facotrs that impact the world gross of a movie?

Modules & Functions

In [86]:
```
%%HTML
<script>
    function luc21893_refresh_cell(cell) {
        if( cell.luc21893 ) return;
        cell.luc21893 = true;
        console.debug('New code cell found...' );

        var div = document.createElement('DIV');
        cell.parentNode.insertBefore( div, cell.nextSibling );
        div.style.textAlign = 'right';
        var a = document.createElement('A');
        div.appendChild(a);
        a.href='#'
        a.luc21893 = cell;
        a.setAttribute( 'onclick', "luc21893_toggle(this); return false;" )

        cell.style.visibility='hidden';
        cell.style.position='absolute';
        a.innerHTML = '[show code]';

    }
    function luc21893_refresh() {
        if( document.querySelector('.code_cell .input') == null ) {
            // it apeears that I am in a exported html
            // hide this code
            var codeCells = document.querySelectorAll('.jp-InputArea')
            codeCells[0].style.visibility = 'hidden';
            codeCells[0].style.position = 'absolute';
            for( var i = 1; i < codeCells.length; i++ ) {
                luc21893_refresh_cell(codeCells[i].parentNode)
            }
            window.onload = luc21893_refresh;
        }
        else {
            // it apperas that I am in a jupyter editor
            var codeCells = document.querySelectorAll('.code_cell .input')
            for( var i = 0; i < codeCells.length; i++ ) {
                luc21893_refresh_cell(codeCells[i])
            }
            window.setTimeout( luc21893_refresh, 1000 )
        }
    }

    function luc21893_toggle(a) {
        if( a.luc21893.style.visibility=='hidden' ) {
            a.luc21893.style.visibility='visible';
            a.luc21893.style.position='';
            a.innerHTML = '[hide code]';
        }
        else {
            a.luc21893.style.visibility='hidden';
            a.luc21893.style.position='absolute';
            a.innerHTML = '[show code]';
        }
    }
```

```
    luc21893_refresh()
</script>
```

In [1]:

```
import re
import json
import requests
import requests_cache
requests_cache.install_cache("STA_141B_Final_Project")
import time
import lxml.html as lx
import numpy as np
import pandas as pd
import seaborn as sns
import plotnine as p9
import statsmodels.api as sm # For Linear Regression ANOVA (WARNING: str()
import statsmodels.stats.api as sms # For Linear Regression BP test
import statsmodels.formula.api as smf # For Linear Regression to have speci
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib.pyplot as plt # For Linear Regression Assumption Graphs
from matplotlib import gridspec # For Linear Regression Assumption Graphs
from IPython.display import display_html # For Linear Regression Assumption
from IPython.display import display, HTML # For Side-By-Side Pandas Datafra
from itertools import chain,cycle # For Linear Regression Assumption Graphs
import scipy # For Logistic Regression LR test
from scipy.stats import norm # For Linear Regression QQ-plot
import plotly.express as px # For Linear Regression Assumption Graphs
import plotly.graph_objects as go # For Linear Regression Assumption Graphs
import plotly.figure_factory as ff # For Distribution Graph
import plotly.io as pio
pio.renderers.default='notebook'
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
def extract_data(number):
    Name,Date,production_bug,Domestic_Gross,Worldwide_Gross=[],[],[],[],[]
    url="https://www.the-numbers.com/movie/budgets/all" #the inital url for
    header_num = {"user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    #This part use web script to extract the imformation from the number we
    response_num = requests.get(url, headers = header_num)
    html_num = lx.fromstring(response_num.text)
    Raw=[x.text_content() for x in html_num.xpath("//table//a")]
    Name.append(Raw[1::2]) #extracts the movie name column
    Date.append(Raw[0::2]) #extracts the release date column
    Rawdata=[float(x.text_content().replace("\xa0$","").replace(",","")) fo
    production_bug.append(Rawdata[1::4]) #append the name data into the emp
    Domestic_Gross.append(Rawdata[2::4]) #append the domestic gross column
    Worldwide_Gross.append(Rawdata[3::4]) #append
    for i in range(101,100*number+2,100):
        url="https://www.the-numbers.com/movie/budgets/all/"+str(i)
        response_num = requests.get(url, headers = header_num)
        html_num = lx.fromstring(response_num.text)
        Raw=[x.text_content() for x in html_num.xpath("//table//a")]
        Rawdata=[float(x.text_content().replace("\xa0$","").replace(",","")
        Name.append(Raw[1::2])
        Date.append(Raw[0::2])
        production_bug.append(Rawdata[1::4])
        Domestic_Gross.append(Rawdata[2::4])
        Worldwide_Gross.append(Rawdata[3::4])
    Date1=list(chain.from_iterable(Date))
    Date2=[item.replace('Unknown', '') for item in Date1]
    Date3=pd.to_datetime(Date2)
    Name1=list(chain.from_iterable(Name))
    production_bug1=list(chain.from_iterable(production_bug))
    Domestic_Gross1=list(chain.from_iterable(Domestic_Gross))
    Worldwide_Gross1=list(chain.from_iterable(Worldwide_Gross))
    dic={"Release Date":Date3,"Movie":Name1,"Production Budget":production_
         "Domestic Gross":Domestic_Gross1,"Worldwide Gross":Worldwide_Gross
    dic_dataframe = pd.DataFrame(dic)
    dic_dataframe['Movie'] = dic_dataframe['Movie'].astype(str) #convert Mo
    dic_dataframe['Production Budget'] = dic_dataframe['Production Budget']
    dic_dataframe['Domestic Gross'] = dic_dataframe['Domestic Gross'].astyp
    dic_dataframe['Worldwide Gross'] = dic_dataframe['Worldwide Gross'].ast
    return dic_dataframe
final6340=extract_data(63)
final6340["Release Date"]=pd.DatetimeIndex(final6340['Release Date']).year
#i just added the raw data into the forloop, so the raw date will update fo
def unicodetoascii(text):
    TEXT = (text.
            replace('\\xe2\\x80\\x99', "'").
            replace('\\xc3\\xa9', 'e').
            replace('\\xe2\\x80\\x90', '-').
            replace('\\xe2\\x80\\x91', '-').
            replace('\\xe2\\x80\\x92', '-').
            replace('\\xe2\\x80\\x93', '-').
            replace('\\xe2\\x80\\x94', '-').
            replace('\\xe2\\x80\\x94', '-').
            replace('\\xe2\\x80\\x98', "'").
            replace('\\xe2\\x80\\x9b', "'").
            replace('\\xe2\\x80\\x9c', '"').
```

```
            replace('\\xe2\\x80\\x9c', '"').
            replace('\\xe2\\x80\\x9d', '"').
            replace('\\xe2\\x80\\x9e', '"').
            replace('\\xe2\\x80\\x9f', '"').
            replace('\\xe2\\x80\\xa6', '...').
            replace('\\xe2\\x80\\xb2', "'").
            replace('\\xe2\\x80\\xb3', "'").
            replace('\\xe2\\x80\\xb4', "'").
            replace('\\xe2\\x80\\xb5', "'").
            replace('\\xe2\\x80\\xb6', "'").
            replace('\\xe2\\x80\\xb7', "'").
            replace('\\xe2\\x81\\xba', "+").
            replace('\\xe2\\x81\\xbb', "-").
            replace('\\xe2\\x81\\xbc', "=").
            replace('\\xe2\\x81\\xbd', "(").
            replace('\\xe2\\x81\\xbe', ")").
            replace('â\x80\x99',"'").
            replace('â\x80¦'," ").
            replace("(ë°\x80ì\xa0\x95)","").
            replace("Ã©","é").
            replace("â\x80\x94"," ").
            replace("Ã´","ô").
            replace("Ã¼","ü").
            replace("Ã¯","i").
            replace("Ã","É").
            replace("É¨","è").
            replace("â\x80\x93","—").
            replace("É","à").
            replace("§","ç").
            replace("Âº","º").
            replace("à³","ó").
            replace("Â","").
            replace("à«","ë").
            replace("à¤","ä").
            replace("à»","û").
            replace("à","É").
            replace("É•p","Ép").
            replace("Éª","ê").
            replace("É£","ã").
            replace("É¥","å").
            replace("Éº","ú").
            replace("É¸","ø").
            replace("É","í"))
    return TEXT
def rare_title_list(list_title_upp, add_apos = True):
    common_character_list = [
    " ",
    "0","1","2","3","4","5","6","7","8","9",
    "A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R"
    if add_apos == True:
        common_character_list.append("'")
    rare_title_list = []
    for title in list_title_upp:
        condition_title = []
        for character in title:
            condition_title.append(character not in common_character_list)
        if sum(condition_title) >= 1:
```

```
                rare_title_list.append(title)
        return rare_title_list
def final_name_change(text):
    TEXT = (text.
            replace("10 000 B C (2008)","10 000 BC (2008)").
            replace("SOUTHLAND TALES (2007)","SOUTHLAND TALES (2006)").
            replace("SPICE WORLD (1998)","SPICE WORLD (1997)").
            replace("SPIDER MAN INTO THE SPIDER VERSE 3D (2018)","SPIDER MA
            replace("SPRING BREAKERS (2013)","SPRING BREAKERS (2012)").
            replace("SPY KIDS 2 ISLAND OF LOST DREAMS (2002)","SPY KIDS 2 T
            replace("SPY KIDS 3 D GAME OVER (2003)","SPY KIDS 3 GAME OVER (
            replace("SPY KIDS 4 ALL THE TIME IN THE WORLD (2011)","SPY KIDS
            replace("TAE GUIK GI THE BROTHERHOOD OF WAR (2004)","TAE GUK GI
            replace("TEETH (2008)","TEETH (2007)").
            replace("THE ADVENTURES OF SHARKBOY AND LAVAGIRL 3 D (2005)","T
            replace("THE BOONDOCK SAINTS (2000)","THE BOONDOCK SAINTS (1999
            replace("THE BROWN BUNNY (2004)","THE BROWN BUNNY (2003)").
            replace("THE GREEN INFERNO (2015)","THE GREEN INFERNO (2013)").
            replace("THE HILLS HAVE EYES 2 (2007)","THE HILLS HAVE EYES II 
            replace("THE INFORMERS (2009)","THE INFORMERS (2008)").
            replace("THE LORDS OF SALEM (2013)","THE LORDS OF SALEM (2012)"
            replace("TWIXT (2012)","TWIXT (2011)").
            replace("WALKING WITH DINOSAURS 3D (2013)","WALKING WITH DINOSA
            replace("MEMENTO (2001)","MEMENTO (2000)").
            replace("MOVIE 43 (2012)","MOVIE 43 (2013)").
            replace("NIGHT ON EARTH (1992)","NIGHT ON EARTH (1991)").
            replace("OUIJA (II) (2014)","OUIJA (2014)").
            replace("POLICE ACADEMY 7 MISSION TO MOSCOW (1994)","POLICE ACA
            replace("PORTRAIT DE LA JEUNE FILLE EN FEU (2019)","PORTRAIT OF
            replace("SHARK NIGHT (2011)","SHARK NIGHT 3D (2011)").
            replace("SILENT HOUSE (2012)","SILENT HOUSE (2011)").
            replace("300 (2007)", "300 (2006)").
            replace("AMORES PERROS (2001)", "AMORES PERROS (2000)").
            replace("AN AMERICAN HAUNTING (2006)", "AN AMERICAN HAUNTING (2
            replace("BLOODRAYNE (2006)","BLOODRAYNE (2005)").
            replace("BOAT TRIP (2003)","BOAT TRIP (2002)").
            replace("CASABLANCA (1943)","CASABLANCA (1942)").
            replace("DARKNESS (2004)","DARKNESS (2002)").
            replace("DAWN OF THE DEAD (1979)","DAWN OF THE DEAD (1978)").
            replace("DOA DEAD OR ALIVE (2007)","DOA DEAD OR ALIVE (2006)").
            replace("DONKEY PUNCH (2009)","DONKEY PUNCH (2008)").
            replace("DYLAN DOG DEAD OF NIGHT (2011)","DYLAN DOG DEAD OF NIG
            replace("EX MACHINA (2015)", "EX MACHINA (2014)").
            replace("EYE OF THE BEHOLDER (2000)", "EYE OF THE BEHOLDER (199
            replace("FATHER'S DAY (1997)", "FATHERS' DAY (1997)").
            replace("FIFTY SHADES FREED (2018)", "FIFTY SHADES DARKER (2017
            replace("HARRY POTTER AND THE DEATHLY HALLOWS PART 1 (2010)", "
            replace("HIGHLANDER THE FINAL DIMENSION (1995)", "HIGHLANDER TH
            replace("IN THE NAME OF THE KING A DUNGEON SIEGE TALE (2008)", 
            replace("JASON X (2002)", "JASON X (2001)").
            replace("KINGSMAN THE SECRET SERVICE (2015)", "KINGSMAN THE SEC
            replace("LOCK STOCK AND TWO SMOKING BARRELS (1999)", "LOCK STOC
                 )
    return TEXT
def meta_score(i):
    return i.text_content().replace("\n","").replace("\xa0","").replace("
def title_genre(desired_genre):
```

```
    return FinalCombine_No_NaN.loc[[True if x == desired_genre else False f
def oscar_indicator_row(df, role, year_of_movie, row):
    if role == "Lead":
        oscar_role = "Oscar_Actor"
    elif role == "Director":
        oscar_role = "Oscar_Director"
    list_of_lists = oscar_df.loc[[True if x < year_of_movie else False for
    one_list = [x for sublist in list_of_lists for x in sublist]
    if type(df[role][row]) != list:
        people_in_row = [df[role][row]]
    else:
        people_in_row = df[role][row]
    condition_list = []
    for i in range(0, len(people_in_row)):
        condition_list.append(people_in_row[i] in one_list)
    if sum(condition_list) >= 1:
        return 1
    else:
        return 0
def highlight_problem_rows(s):
    return 'background-color: %s' % 'red'
def display_side_by_side(dfs:list, captions:list):
    output = ""
    combined = dict(zip(captions, dfs))
    styles = [dict(selector="caption", props=[("font-size", "150%"),("color
    for caption, df in combined.items():
        output += df.set_table_attributes("style='display:inline'").set_cap
        output += "\xa0\xa0\xa0"
    display(HTML(output))
def lin_assump(f, df, res = False, qqp = False, lev = False, coef = False,
    ols_fit = smf.ols(formula = f, data = df).fit()
    results_as_html1 = ols_fit.summary().tables[1].as_html()
    coef_pval_df = pd.read_html(results_as_html1, header=0, index_col=0)[0]
    insig_rows = [x for x,y in zip(coef_pval_df.index,coef_pval_df["P>|t|"]
    coef_pval_df_highlight = coef_pval_df.style.applymap(highlight_problem_
    outliers_df = pd.DataFrame({"abs_res": abs(ols_fit.resid_pearson)},inde
    outliers_df = outliers_df[outliers_df["abs_res"] > 3].sort_values(by=["
    outliers_df_highlight = outliers_df.style.applymap(highlight_problem_ro
    results_as_html2 = ols_fit.summary().tables[2].as_html()
    dw_stat = round(float(pd.read_html(results_as_html2, header=0, index_co
    if dw_stat > 1.5 and dw_stat < 2.5:
        dw_conclude = "NOT CORRELATED"
    else:
        dw_conclude = "PROBLEM: CORRELATED"
    dw_row = [dw_stat,dw_conclude]
    results_as_html0 = ols_fit.summary().tables[0].as_html()
    r_sq_adj = pd.read_html(results_as_html0, header=0, index_col=0)[0].ilo
    if r_sq_adj >= 0.5:
        r_sq_adj_conclude = "MODEL EXPLAIN AT LEAST HALF VAR"
    else:
        r_sq_adj_conclude = "PROBLEM: MODEL EXPLAIN LESS THAN HALF VAR"
    r_sq_adj_row = [r_sq_adj,r_sq_adj_conclude]
    stat_df = pd.DataFrame([dw_row,r_sq_adj_row], index = ["error_corr_dw",
    problem_rows = [x for x,y in zip(stat_df.index,stat_df["conclude"]) if
    stat_df_highlight = stat_df.style.applymap(highlight_problem_rows, subs
    output_var = f.split(" ~")[0]
    input_var = f.replace(output_var + " ~ ","").split(" + ")
```

```
    df_in = df[input_var]
    df_in_dummies = pd.get_dummies(df_in)
    vif_data = pd.DataFrame()
    vif_data["feature"] = df_in_dummies.columns
    vif_data["VIF"] = [variance_inflation_factor(df_in_dummies.values, i) f
    vif_data = vif_data.set_index("feature")
    vif_data.index.name = None
    vif_data = vif_data.sort_values(by = "VIF", ascending = False)
    VIF_problem_rows = [x for x,y in zip(vif_data.index,vif_data["VIF"]) if
    vif_data_highlight = vif_data.style.applymap(highlight_problem_rows, su
    fit_res_plot = (p9.ggplot(df,mapping = p9.aes(x = 'ols_fit.fittedvalues
        + p9.geom_point()
        + p9.labs(title = "Fitted vs Residuals", x = 'Fitted values', y = '
        + p9.geom_text(p9.aes(label = [x if abs(y) > 3 else "" for x,y in z
        + p9.theme_bw())
    res_quant_df = pd.DataFrame({"res": ols_fit.resid_pearson}, index = df.
    res_quant_df = res_quant_df.sort_values(by = ["res"])
    fi = [(i - 0.5) / len(ols_fit.resid_pearson) for i in range(1,len(ols_f
    res_quant_df["quant"] = [norm.ppf(x) for x in fi]
    QQ_plot = (p9.ggplot(res_quant_df,mapping = p9.aes(x = "quant", y = "re
        + p9.geom_point()
        + p9.geom_abline(p9.aes(intercept = 0, slope = 1), color = 'blue')
        + p9.labs(title = "QQ-Plot with Pearson Residuals")
        + p9.geom_text(p9.aes(label = [x if abs(y) > 3 else "" for x,y in z
        + p9.theme_bw())
    lev_res_plot = (p9.ggplot(df,mapping = p9.aes(x = 'ols_fit.get_influenc
        + p9.geom_point()
        + p9.labs(title = "Leverage vs Residuals", x = 'Leverage', y = 'Res
        + p9.geom_text(p9.aes(label = [x if abs(y) > 3 else "" for x,y in z
        + p9.theme_bw())
    if res == True:
        res_df = pd.DataFrame(index = df.index)
        res_df["fit_val"] = ols_fit.fittedvalues
        res_df["res"] = ols_fit.resid_pearson
        fit_res_fig = px.scatter(res_df, x = "fit_val", y = "res",
                                 labels = {"fit_val": "Fitted Values",
                                           "res": "Residuals"},
                                 title = "Fitted vs Residuals",
                                 color_discrete_sequence = ["black"])
        fit_res_fig.add_hline(y = 3, line_width = 1.5, line_dash = "dash",
        fit_res_fig.add_hrect(y0 = 3, y1 = 100, fillcolor="red", opacity=0.
        fit_res_fig.add_hline(y = -3, line_width = 1.5, line_dash = "dash",
        fit_res_fig.add_hrect(y0 = -100, y1 = -3, fillcolor="red", opacity=
        fit_res_fig.add_trace(go.Scatter(
            x = [x for x,y in zip(res_df["fit_val"], res_df["res"]) if abs(
            y = [y for y in res_df["res"] if abs(y) > 3],
            mode = "markers+text",
            name = "Outlier",
            text = [x for x,y in zip(res_df.index, res_df["res"]) if abs(y)
            textfont={"color":"red"},
            textposition = "bottom center",
            marker = {"color": "red"}))
        fit_res_fig.update_xaxes(showgrid=True, gridwidth=1.5, gridcolor='w
        fit_res_fig.update_yaxes(showgrid=True, gridwidth=1.5, gridcolor='w
        fit_res_fig.update_layout({
            'plot_bgcolor': 'rgb(235,235,235)'})
        fit_res_fig.show()
```

```
    elif qqp == True:
        res_quant_df = pd.DataFrame(index = df.index)
        res_quant_df["res"] = ols_fit.resid_pearson
        res_quant_df = res_quant_df.sort_values(by = ["res"])
        fi = [(i - 0.5) / len(ols_fit.resid_pearson) for i in range(1,len(o
        res_quant_df["quant"] = [norm.ppf(x) for x in fi]
        res_quant_fig = px.scatter(res_quant_df, x = "quant", y = "res",
                                   labels = {"quant": "Theoretical Quantile
                                             "res": "Sample Quantiles"},
                                   title = "QQ-Plot with Pearson Residuals"
                                   color_discrete_sequence = ["black"])
        res_quant_fig.add_trace(go.Scatter(
            x = [-100,100],
            y = [-100,100],
            name = "Normality Line",
            line_color = "blue"))
        res_quant_fig.add_trace(go.Scatter(
            x = [x for x,y in zip(res_quant_df["quant"], res_quant_df["res"
            y = [y for y in res_quant_df["res"] if abs(y) > 3],
            mode = "markers+text",
            name = "Outlier",
            text = [x for x,y in zip(res_quant_df.index, res_quant_df["res"
            textfont={"color":"red"},
            textposition = "bottom center",
            marker = {"color": "red"}))
        res_quant_fig.update_xaxes(showgrid=True, gridwidth=1.5, gridcolor=
        res_quant_fig.update_yaxes(showgrid=True, gridwidth=1.5, gridcolor=
        res_quant_fig.update_layout({
            'plot_bgcolor': 'rgb(235,235,235)'})
        res_quant_fig.show()
    elif lev == True:
        lev_df = pd.DataFrame(index = df.index)
        lev_df["lev"] = ols_fit.get_influence().hat_matrix_diag
        lev_df["res"] = ols_fit.resid_pearson
        lev_res_fig = px.scatter(lev_df, x = "lev", y = "res",
                                 labels = {"lev": "Leverage",
                                           "res": "Residuals"},
                                 title = "Leverage vs Residuals",
                                 color_discrete_sequence = ["black"])
        lev_res_fig.add_hline(y = 3, line_width = 1.5, line_dash = "dash",
        lev_res_fig.add_hrect(y0 = 3, y1 = 100, fillcolor="red", opacity=0.
        lev_res_fig.add_hline(y = -3, line_width = 1.5, line_dash = "dash",
        lev_res_fig.add_hrect(y0 = -100, y1 = -3, fillcolor="red", opacity=
        lev_res_fig.add_trace(go.Scatter(
            x = [x for x,y in zip(lev_df["lev"], lev_df["res"]) if abs(y) >
            y = [y for y in lev_df["res"] if abs(y) > 3],
            mode = "markers+text",
            name = "Outlier",
            text = [x for x,y in zip(lev_df.index, lev_df["res"]) if abs(y)
            textfont={"color":"red"},
            textposition = "bottom center",
            marker = {"color": "red"}))
        lev_res_fig.update_xaxes(showgrid=True, gridwidth=1.5, gridcolor='w
        lev_res_fig.update_yaxes(showgrid=True, gridwidth=1.5, gridcolor='w
        lev_res_fig.update_layout({
            'plot_bgcolor': 'rgb(235,235,235)'})
        lev_res_fig.show()
```

```
    elif coef == True:
        display(coef_pval_df_highlight)
    elif vif == True:
        display(vif_data_highlight)
    else:
        coef_pval_df_core = pd.read_html(results_as_html1, header=0, index_
        coef_pval_df_core_highlight = coef_pval_df_core.style.applymap(high
        p1 = fit_res_plot
        p2 = QQ_plot
        p3 = lev_res_plot
        fig = (p9.ggplot()+p9.geom_blank(data=df)+p9.theme_void()+p9.theme(
        gs = gridspec.GridSpec(nrows = 1, ncols = 3)
        ax1 = fig.add_subplot(gs[0,0])
        ax2 = fig.add_subplot(gs[0,1])
        ax3 = fig.add_subplot(gs[0,2])
        ax1.title.set_text('Fitted vs Residuals')
        ax2.title.set_text('QQ-Plot with Pearson Residuals')
        ax3.title.set_text('Leverage vs Residuals')
        p1._draw_using_figure(fig, [ax1])
        p2._draw_using_figure(fig, [ax2])
        p3._draw_using_figure(fig, [ax3])
        plt.show()
        display_side_by_side([coef_pval_df_core_highlight, outliers_df_high
def model_red(df, model = "linear", outvar = None, invar = None, f = None,
    if model == "linear":
        ols_fit = smf.ols(f, data = df).fit()
        results_as_html1 = ols_fit.summary().tables[1].as_html()
        coef_pval_df = pd.read_html(results_as_html1, header=0, index_col=0
        insig_rows = [x for x,y in zip(coef_pval_df.index,coef_pval_df["P>|
        coef_pval_df_highlight = coef_pval_df.style.applymap(highlight_prob
        output_var = f.split(" ~")[0]
        input_var = f.replace(output_var + " ~ ","").split(" + ")
        df_in = df[input_var]
        df_in_dummies = pd.get_dummies(df_in)
        vif_data = pd.DataFrame()
        vif_data["feature"] = df_in_dummies.columns
        vif_data["VIF"] = [variance_inflation_factor(df_in_dummies.values,
        vif_data = vif_data.set_index("feature")
        vif_data.index.name = None
        vif_data = vif_data.sort_values(by = "VIF", ascending = False)
        VIF_problem_rows = [x for x,y in zip(vif_data.index,vif_data["VIF"]
        vif_data_highlight = vif_data.style.applymap(highlight_problem_rows
        results_as_html0 = ols_fit.summary().tables[0].as_html()
        aic_bic = pd.read_html(results_as_html0, header=0, index_col=0)[0].
        aic_bic = [x for x in aic_bic]
        aic_bic_df = pd.DataFrame(index = ["AIC", "BIC"])
        aic_bic_df["stat"] = aic_bic
        if rm == None:
            display_side_by_side([coef_pval_df_highlight, vif_data_highligh
        else:
            remove_var_minus = ["-" + x for x in rm]
            string = ""
            for x in remove_var_minus:
                string += x
            ols_fit_red = smf.ols(f + string, data = df).fit()
            results_as_html1_red = ols_fit_red.summary().tables[1].as_html(
            coef_pval_df_red = pd.read_html(results_as_html1_red, header=0,
```

```
            insig_rows_red = [x for x,y in zip(coef_pval_df_red.index,coef_
            coef_pval_df_highlight_red = coef_pval_df_red.style.applymap(hi
            input_var_red = input_var.copy()
            for i in rm:
                input_var_red.remove(i)
            df_in_red = df[input_var_red]
            df_in_dummies_red = pd.get_dummies(df_in_red)
            vif_data_red = pd.DataFrame()
            vif_data_red["feature"] = df_in_dummies_red.columns
            vif_data_red["VIF"] = [variance_inflation_factor(df_in_dummies_
            vif_data_red = vif_data_red.set_index("feature")
            vif_data_red.index.name = None
            vif_data_red = vif_data_red.sort_values(by = "VIF", ascending =
            VIF_problem_rows_red = [x for x,y in zip(vif_data_red.index,vif
            vif_data_highlight_red = vif_data_red.style.applymap(highlight_
            results_as_html0_red = ols_fit_red.summary().tables[0].as_html(
            aic_bic_red = pd.read_html(results_as_html0_red, header=0, inde
            aic_bic_red = [x for x in aic_bic_red]
            aic_bic_df_red = pd.DataFrame(index = ["AIC", "BIC"])
            aic_bic_df_red["stat"] = aic_bic_red
            display_side_by_side([coef_pval_df_highlight, vif_data_highligh
    elif model == "logit":
        df_logit = sm.add_constant(df)
        const_invar = ["const"]
        for i in invar:
            const_invar.append(i)
        logit_fit = sm.GLM(df_logit[outvar], df_logit[const_invar], family=
        coef_pval_df_logit = pd.read_html(logit_fit.summary().tables[1].as_
        insig_rows_logit = [x for x,y in zip(coef_pval_df_logit.index,coef_
        coef_pval_df_highlight_logit = coef_pval_df_logit.style.applymap(hi
        df_in = df[invar]
        df_in_dummies = pd.get_dummies(df_in)
        vif_data = pd.DataFrame()
        vif_data["feature"] = df_in_dummies.columns
        vif_data["VIF"] = [variance_inflation_factor(df_in_dummies.values,
        vif_data = vif_data.set_index("feature")
        vif_data.index.name = None
        vif_data = vif_data.sort_values(by = "VIF", ascending = False)
        VIF_problem_rows = [x for x,y in zip(vif_data.index,vif_data["VIF"]
        vif_data_highlight = vif_data.style.applymap(highlight_problem_rows
        if rm == None:
            display_side_by_side([coef_pval_df_highlight_logit, vif_data_hi
        else:
            const_invar_red = const_invar.copy()
            for i in rm:
                const_invar_red.remove(i)
            logit_fit_red = sm.GLM(df_logit[outvar], df_logit[const_invar_r
            coef_pval_df_logit_red = pd.read_html(logit_fit_red.summary().t
            insig_rows_logit_red = [x for x,y in zip(coef_pval_df_logit_red
            coef_pval_df_highlight_logit_red = coef_pval_df_logit_red.style
            invar_red = invar.copy()
            for i in rm:
                invar_red.remove(i)
            df_in_red = df[invar_red]
            df_in_dummies_red = pd.get_dummies(df_in_red)
            vif_data_red = pd.DataFrame()
            vif_data_red["feature"] = df_in_dummies_red.columns
```

```
                vif_data_red["VIF"] = [variance_inflation_factor(df_in_dummies_
                vif_data_red = vif_data_red.set_index("feature")
                vif_data_red.index.name = None
                vif_data_red = vif_data_red.sort_values(by = "VIF", ascending =
                VIF_problem_rows_red = [x for x,y in zip(vif_data_red.index,vif
                vif_data_highlight_red = vif_data_red.style.applymap(highlight_
                display_side_by_side([coef_pval_df_highlight_logit, vif_data_hi
```

The cells above show the modules and functions that we used for our analysis. Most of them were used for data collection purposes (i.e. constructing data frames, web scraping websites, standardizing movie titles). Meanwhile, some were simply used for technical formatting (i.e. hiding warning messages, highlighting insignificant variables, displaying tables side by side). Functions such as lin_assump() and model_red() were used to check linear regression assumptions and to reduce generalized linear models respectively.

Data Collection

In [ ]:
```
final6340["Movie"]=[unicodetoascii(x) for x in final6340["Movie"]]
final6340[final6340["Domestic Gross"]==0] #no zero
finalno0=final6340[(final6340['Domestic Gross'] !=0) & (final6340['Worldwid
finalno=finalno0[finalno0["Release Date"].notnull()].reset_index(drop=True)
finalno["Release Date"]=finalno["Release Date"].astype(np.int64) #convert t
movie_tit = [x.upper() #this list keep tracks of the edits for the movie ti
             .replace(" : "," ")
             .replace(": "," ")
             .replace(" :"," ")
             .replace(":"," ")
             .replace(" . "," ")
             .replace(". "," ")
             .replace(" ."," ")
             .replace("."," ")
             .replace(" & "," AND ")
             .replace("& "," AND ")
             .replace(" &"," AND ")
             .replace("&"," AND ")
             .replace(" - "," ")
             .replace(" -"," ")
             .replace("- "," ")
             .replace("-"," ")
             .replace("É","E")
             .replace(" \\ ","")
             .replace("\\ ","")
             .replace(" \\","")
             .replace("\\","")
             .replace("   "," ")
             .replace(" , "," ")
             .replace(" ,"," ")
             .replace(", "," ")
             .replace(","," ")
             .replace(" ! "," ")
             .replace(" !"," ")
             .replace("! "," ")
             .replace("!"," ")
             .replace(" ? "," ")
             .replace(" ?"," ")
             .replace("? "," ")
             .replace("?"," ")
             .replace("Ü","U")
             .replace("Ë","E")
             .replace("È","E")
             .replace("É","E")
             .replace("È","E")
             .replace("Ä","A")
             .replace("Á","A")
             .replace("À","A")
             .replace("Í","I")
             .replace("Ç","C")
             .replace("Ô","O")
             .replace(" / "," ")
             .replace(" /"," ")
             .replace("/ "," ")
             .replace("/"," ")
             .replace("¢","C")
```

```
                .replace(" ( "," ")
                .replace("( "," ")
                .replace(" ("," ")
                .replace("("," ")
                .replace(" ) "," ")
                .replace(" )"," ")
                .replace(") "," ")
                .replace(")"," ")
                .replace(" [ "," ")
                .replace("[ "," ")
                .replace(" ["," ")
                .replace("["," ")
                .replace(" ] "," ")
                .replace(" ]"," ")
                .replace("] "," ")
                .replace("]"," ")
                .replace(" * "," ")
                .replace(" *"," ")
                .replace("* "," ")
                .replace("*"," ")
                .replace(" + "," AND ")
                .replace(" +"," AND ")
                .replace("+ "," AND ")
                .replace("+"," AND ")
                .replace(" ¦ "," ")
                .replace("¦ "," ")
                .replace(" ¦"," ")
                .replace("¦"," ")
                .replace(" # "," ")
                .replace(" #"," ")
                .replace("# "," ")
                .replace("#"," ")
                .replace(" º "," ")
                .replace(" º"," ")
                .replace("º "," ")
                .replace(" º "," ")
                .replace(" ¹ "," ")
                .replace("¹ "," ")
                .replace(" ¹"," ")
                .replace("¹"," ")
                .replace('THE CHRONICLES OF NARNIA THE LION THE WITCH A…','THE
                .replace('THE CHRONICLES OF NARNIA THE VOYAGE OF THE DAW…','TH
                .replace('PIRATES OF THE CARIBBEAN THE CURSE OF THE BLAC…','PI
                .replace('BIRDS OF PREY AND THE FANTABULOUS EMANCIPATION…','BI
                .replace('THE ASSASSINATION OF JESSE JAMES BY THE COWARD …','T
                .replace('ALEXANDER AND THE TERRIBLE HORRIBLE NO GOOD …','ALEX
                .replace('TEENAGE MUTANT NINJA TURTLES II THE SECRET OF …','TE
                .replace("THE PIRATES WHO DON'T DO ANYTHING A VEGGIETALE…","TH
                .replace('HANNAH MONTANA AND MILEY CYRUS BEST OF BOTH WO…','HA
                .replace("HILLARY'S AMERICA THE SECRET HISTORY OF THE DE…","ST
                .replace("A NIGHTMARE ON ELM STREET PART 2 FREDDY'S REVE…","A 
                .replace("MARILYN HOTCHKISS' BALLROOM DANCING AND CHARM S…","M
                .replace('POM WONDERFUL PRESENTS THE GREATEST MOVIE EVER…',"PO
                .replace('ONCE IN A LIFETIME THE EXTRAORDINARY STORY OF …','ON
                .replace('AQUA TEEN HUNGER FORCE COLON MOVIE FILM FOR THE…','A
                .replace('DECEPTIVE PRACTICE THE MYSTERIES AND MENTORS O…','DE
                .replace('NÃO PARE NA PISTA A MELHOR HISTORIA DE PAULO C…',"NA
```

```
              .replace('CI¬KE NIE YINNIIING','CI KE NIE YIN NIANG')
              .replace('Ó',"O")
              .replace('¡',"I")
              .replace('Ú','U')
              .replace('Ê',"E")
              .replace('I\xad',"I")
              .replace('Å',"A")
              .replace('Û',"U")
              .replace('I\xa0',"A")
             for x in finalno["Movie"]]
rare_title_list(movie_tit)
removie_list = rare_title_list(movie_tit)
#this removie list shows the movies we want to filter out
#[x for x in movie_tit if x.startswith(" ") or x.endswith(" ")] #replace an
movie_tit = [x.rstrip(" ") if x.endswith(" ") else x for x in movie_tit]
movie_tit = [x.lstrip(" ") if x.startswith(" ") else x for x in movie_tit]
clean_movie_tit = pd.DataFrame(movie_tit) #clean movie title
#clean_movie_tit
tem_dataset = pd.concat([finalno,clean_movie_tit],axis=1)
tem_dataset.columns = [*tem_dataset.columns[:-1], 'Clean Movie Name']
desire_list = tem_dataset.Movie.isin(removie_list)
#desire_list
tem_dataset = pd.concat([finalno,clean_movie_tit],axis=1)
tem_dataset.columns = [*tem_dataset.columns[:-1], 'Clean Movie Name']
tem_dataset = tem_dataset.loc[[False if x in removie_list else True for x i
tem_dataset = tem_dataset.sort_values(by = "Clean Movie Name")
tem_dataset["Movie"] = [x + " (" + str(y) + ")" for x,y in zip(tem_dataset[
tem_dataset = tem_dataset.iloc[:,[1,2,3,4]]
tem_dataset = tem_dataset.reset_index(drop=True)
tem_dataset
#tem_dataset.head(20)
#tem_dataset.to_csv("Completed_Dataset_Part1.csv")
first_half = 'https://www.imdb.com/search/title/?groups=top_1000&sort=boxof
last_half = "&ref_=adv_nxt"
first_half = 'https://www.imdb.com/search/title/?groups=bottom_1000&sort=bo
last_half = "&ref_=adv_prv"
#1-20pages
first_half = 'https://www.imdb.com/search/title/?groups=top_1000&sort=boxof
last_half = "&ref_=adv_nxt"
title = []
genre = []
movie_rating = []
user_rating = []
metascore = []
duration = []
director = []
lead = []
year = []
titlefinal = []
genrefinal = []
moviefinal = []
userfinal = []
metafinal = []
durationfinal = []
directorfinal = []
leadfinal = []
yearfinal = []
```

```
TitleData = pd.DataFrame()
GenreData= pd.DataFrame()
MovieData = pd.DataFrame()
UserData = pd.DataFrame()
MetaData = pd.DataFrame()
DurationData = pd.DataFrame()
DirectorData = pd.DataFrame()
LeadData = pd.DataFrame()
YearData = pd.DataFrame()
for i in range(1,1001, 50):
    website = first_half + str(i) + last_half
    response_imdb = requests.get(website)
    html_imdb = lx.fromstring(response_imdb.text)
    #For lead and director
    soup = bs(response_imdb.text, 'html.parser')
    for i in range(0,len(soup.find_all("p", class_ = ""))):
        director_stars = soup.find_all("p", class_ = "")[i].get_text().repl
        director.append(director_stars.split("*")[0].split(", "))
        lead.append(director_stars.split("*")[1].split(", ")[0])
    title_page_i = [x.text_content() for x in html_imdb.xpath("//h3//a")]
    year_page_i = [x.text_content().replace('(I) ','') for x in html_imdb.x
    #title_year_page_i = [x + " " + y for x,y in zip(title_page_i,year_page
    year.append(year_page_i)
    title.append(title_page_i)
    genre.append([x.text_content().replace("\n","").replace("            ",
    #movie_rating.append([x.text_content() for x in html_imdb.xpath("//span
    movie_rating.append([x.text_content().replace("\n            ","").repl
    user_rating.append([float(x) for x in html_imdb.xpath("//div[@name = 'i
    list_meta_score = [meta_score(x) if not ("X" in meta_score(x)) else Non
    metascore.append([float(x) if x != None else x for x in list_meta_score
    duration.append([float(x.text_content().replace("\n","").replace("
for i in range(20):#create a for loop and i in range 20
    titlefinal += title[i]
    genrefinal += genre[i]
    moviefinal += movie_rating[i]
    userfinal += user_rating[i]
    metafinal += metascore[i]
    durationfinal += duration[i]
    yearfinal += year[i]
TitleData['Title'] = titlefinal
GenreData['Genre'] = genrefinal
MovieData['Movie Rating'] = moviefinal
UserData['User Rating'] = userfinal
MetaData['Meta Score'] = metafinal
DurationData['Duration in mins'] = durationfinal
DirectorData['Director'] = director
LeadData['Lead'] = lead
YearData['Year'] = yearfinal
combineTop = pd.concat([TitleData,GenreData,MovieData, UserData, MetaData,
#1-20pages
first_half = 'https://www.imdb.com/search/title/?groups=bottom_1000&sort=bo
last_half = "&ref_=adv_prv"
title = []
genre = []
movie_rating = []
user_rating = []
metascore = []
```

```
duration = []
director = []
lead = []
year = []
titlefinal = []
genrefinal = []
moviefinal = []
userfinal = []
metafinal = []
durationfinal = []
directorfinal = []
leadfinal = []
yearfinal = []
TitleData = pd.DataFrame()
GenreData= pd.DataFrame()
MovieData = pd.DataFrame()
UserData = pd.DataFrame()
MetaData = pd.DataFrame()
DurationData = pd.DataFrame()
DirectorData = pd.DataFrame()
LeadData = pd.DataFrame()
YearData = pd.DataFrame()
for i in range(1,1001, 50):
    website = first_half + str(i) + last_half
    response_imdb = requests.get(website)
    html_imdb = lx.fromstring(response_imdb.text)
    #For lead and director
    soup = bs(response_imdb.text, 'html.parser')
    for i in range(0,len(soup.find_all("p", class_ = ""))):
        director_stars = soup.find_all("p", class_ = "")[i].get_text().repl
        director.append(director_stars.split("*")[0].split(", "))
        lead.append(director_stars.split("*")[1].split(", ")[0])
    title_page_i = [x.text_content() for x in html_imdb.xpath("//h3//a")]
    year_page_i = [x.text_content().replace('(I) ','') for x in html_imdb.x
    #title_year_page_i = [x + " " + y for x,y in zip(title_page_i,year_page
    year.append(year_page_i)
    title.append(title_page_i)
    genre.append([x.text_content().replace("\n","").replace("            ",
    #movie_rating.append([x.text_content() for x in html_imdb.xpath("//span
    movie_rating.append([x.text_content().replace("\n            ","").repl
    user_rating.append([float(x) for x in html_imdb.xpath("//div[@name = 'i
    list_meta_score = [meta_score(x) if not ("X" in meta_score(x)) else Non
    metascore.append([float(x) if x != None else x for x in list_meta_score
    duration.append([float(x.text_content().replace("\n","").replace("
for i in range(20):#create a for loop and i in range 20
    titlefinal += title[i]
    genrefinal += genre[i]
    moviefinal += movie_rating[i]
    userfinal += user_rating[i]
    metafinal += metascore[i]
    durationfinal += duration[i]
    yearfinal += year[i]
TitleData['Title'] = titlefinal
GenreData['Genre'] = genrefinal
MovieData['Movie Rating'] = moviefinal
UserData['User Rating'] = userfinal
MetaData['Meta Score'] = metafinal
```

```
DurationData['Duration in mins'] = durationfinal
DirectorData['Director'] = director
LeadData['Lead'] = lead
YearData['Year'] = yearfinal
combineBottom = pd.concat([TitleData,GenreData,MovieData, UserData, MetaDat
FinalCombine = pd.concat([combineTop, combineBottom], axis = 0).reset_index
Userlist = []
for x in FinalCombine['User Rating']:
    if x >= 7:
        Userlist.append('1')
    elif x < 7:
        Userlist.append('0')
    else:
        Userlist.append(None)
Metalist = []
for x in FinalCombine['Meta Score']:
    if x >= 7:
        Metalist.append('1')
    elif x < 7:
        Metalist.append('0')
    else:
        Metalist.append(None)
FinalCombine['User Rating Good/Bad']=Userlist
FinalCombine['Meta Score Good/Bad']=Metalist
FinalCombine_No_NaN = FinalCombine.dropna().reset_index(drop=True) # Final
DirectorOne = []
for x in FinalCombine_No_NaN['Director']:
    if len(x) == 1:
        DirectorOne.append('1')
    else:
        DirectorOne.append('0')
FinalCombine_No_NaN['Director Count 0/1'] = DirectorOne
FinalCombine_No_NaN['Director Count'] = [len(x) for x in FinalCombine_No_Na
#FinalCombine_No_NaN
FinalCombine_No_NaN['Genre'].value_counts().keys() #genres
n = FinalCombine_No_NaN['Genre'].value_counts() #freq
DataGenre= pd.DataFrame(n)
DataGenre = DataGenre.reset_index()
DataGenre.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, in
df_genre_freq = DataGenre.loc[[True if "Horror" in x else False for x in Da
list_to_horror = [df_genre_freq["Genre names"][index] for index in
 [5,7,26,36,49,52,62,63,77,92,94,97,99,100,101,118,130,135,139,140,163,186,
FinalCombine_No_NaN_to_horror = FinalCombine_No_NaN.copy()
FinalCombine_No_NaN_to_horror["Genre"] = ["Horror" if x in list_to_horror e
## Comedy
n1 = FinalCombine_No_NaN_to_horror['Genre'].value_counts() #freq
DataGenre1 = pd.DataFrame(n1)
DataGenre1 = DataGenre1.reset_index()
DataGenre1.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq1 = DataGenre1.loc[[True if "Comedy" in x else False for x in
list_to_h_comedy = [df_genre_freq1["Genre names"][index] for index in
 [3,19,21,27,31,45,62,73,74,79,80,85,101,104,114,115,118,120,126,134,140,14
FinalCombine_No_NaN_to_h_comedy = FinalCombine_No_NaN_to_horror.copy()
FinalCombine_No_NaN_to_h_comedy["Genre"] = ["Comedy" if x in list_to_h_come
## Animation
n2 = FinalCombine_No_NaN_to_h_comedy['Genre'].value_counts() #freq
DataGenre2 = pd.DataFrame(n2)
```

```
DataGenre2 = DataGenre2.reset_index()
DataGenre2.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq2 = DataGenre2.loc[[True if "Animation" in x else False for x
list_to_h_c_ani = [df_genre_freq2["Genre names"][index] for index in
 [3, 21,48,54,84,89,95,111,119,145,162,176,180,186,188,189,190]]
FinalCombine_No_NaN_to_h_c_ani = FinalCombine_No_NaN_to_h_comedy.copy()
FinalCombine_No_NaN_to_h_c_ani["Genre"] = ["Animation" if x in list_to_h_c_
## Drama
n3 = FinalCombine_No_NaN_to_h_c_ani['Genre'].value_counts() #freq
DataGenre3 = pd.DataFrame(n3)
DataGenre3 = DataGenre3.reset_index()
DataGenre3.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq3 = DataGenre3.loc[[True if "Drama" in x else False for x in D
list_to_h_c_a_d = [df_genre_freq3["Genre names"][index] for index in
 [3, 12, 13,14,16,19,21,25,33,35,44,46,52,57,59,60,61,67,68,72,82,93,100,10
FinalCombine_No_NaN_to_h_c_a_d = FinalCombine_No_NaN_to_h_c_ani.copy()
FinalCombine_No_NaN_to_h_c_a_d["Genre"] = ["Drama" if x in list_to_h_c_a_d
n4 = FinalCombine_No_NaN_to_h_c_a_d['Genre'].value_counts() #freq
DataGenre4 = pd.DataFrame(n4)
DataGenre4 = DataGenre4.reset_index()
DataGenre4.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq4 = DataGenre4.loc[[True if "Romance" in x else False for x in
list_to_h_c_a_d_r = [df_genre_freq4["Genre names"][index] for index in
 [4,6,8,23,25,34,35,39,42,43,52,62,65,68,70,78,80,82,92,94,101,102,107,112,
FinalCombine_No_NaN_to_h_c_a_d_r = FinalCombine_No_NaN_to_h_c_a_d.copy()
FinalCombine_No_NaN_to_h_c_a_d_r["Genre"] = ["Romance" if x in list_to_h_c_
n5 = FinalCombine_No_NaN_to_h_c_a_d_r['Genre'].value_counts() #freq
DataGenre5 = pd.DataFrame(n5)
DataGenre5 = DataGenre5.reset_index()
DataGenre5.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq5 = DataGenre5.loc[[True if "Biography" in x else False for x
list_to_h_c_a_d_r_ab = [df_genre_freq5["Genre names"][index] for index in
 [12, 15, 17, 32, 35, 44, 52, 84, 88]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab = FinalCombine_No_NaN_to_h_c_a_d_r.copy
FinalCombine_No_NaN_to_h_c_a_d_r_ab["Genre"] = ["Biography" if x in list_to
n6 = FinalCombine_No_NaN_to_h_c_a_d_r_ab['Genre'].value_counts() #freq
DataGenre6 = pd.DataFrame(n6)
DataGenre6 = DataGenre6.reset_index()
DataGenre6.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq6 = DataGenre6.loc[[True if "Sci-Fi" in x else False for x in
list_to_h_c_a_d_r_ab_sf = [df_genre_freq6["Genre names"][index] for index i
 [19,22,25,27,35,40,51,65,68,72,73,76,77,88]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sf = FinalCombine_No_NaN_to_h_c_a_d_r_a
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sf["Genre"] = ["Sci-fi/Fantasy" if x in
#Fantasy
n7 = FinalCombine_No_NaN_to_h_c_a_d_r_ab_sf['Genre'].value_counts() #freq
DataGenre7 = pd.DataFrame(n7)
DataGenre7 = DataGenre7.reset_index()
DataGenre7.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq7 = DataGenre7.loc[[True if "Fantasy" in x else False for x in
list_to_h_c_a_d_r_ab_sff = [df_genre_freq7["Genre names"][index] for index
 [6,49,65,81,85,87]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff = FinalCombine_No_NaN_to_h_c_a_d_r_
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff["Genre"] = ["Sci-fi/Fantasy" if x i
n8 = FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff['Genre'].value_counts() #freq
DataGenre8 = pd.DataFrame(n8)
DataGenre8 = DataGenre8.reset_index()
```

```
DataGenre8.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq8 = DataGenre8.loc[[True if "Action" in x else False for x in
list_to_h_c_a_d_r_ab_sff_a = [df_genre_freq8["Genre names"][index] for inde
 [7, 8, 9, 12, 13, 15, 17, 18, 20, 21, 24, 30, 32, 35, 36, 39, 40, 41, 43,
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_a = FinalCombine_No_NaN_to_h_c_a_d_
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_a["Genre"] = ["Action/Adventure" if
n9 = FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_a['Genre'].value_counts() #fre
DataGenre9 = pd.DataFrame(n9)
DataGenre9 = DataGenre9.reset_index()
DataGenre9.rename(columns = {'index':'Genre names', 'Genre':'Frequency'}, i
df_genre_freq9 = DataGenre9.loc[[True if "Adventure" in x else False for x
list_to_h_c_a_d_r_ab_sff_aa = [df_genre_freq9["Genre names"][index] for ind
 [1,12,17,19,22,27,28,29,30,31,34,35,42,43,50,52 ]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa = FinalCombine_No_NaN_to_h_c_a_d
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa["Genre"] = ["Action/Adventure" i
#FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_a['Genre'].value_counts() #freq
DataGenre10 = pd.DataFrame(FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa['Genr
DataGenre10 = DataGenre10.reset_index()
DataGenre10.rename(columns = {'index':'Genre names', 'Genre':'Frequency'},
df_genre_freq10 = DataGenre10.loc[[True if "Thriller" in x else False for x
list_to_h_c_a_d_r_ab_sff_aa_t = [df_genre_freq10["Genre names"][index] for
 [8, 15, 18, 21, 24, 26, 27, 31, 35, 36, 37]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t = FinalCombine_No_NaN_to_h_c_a
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t["Genre"] = ["Horror" if x in l
DataGenre11 = pd.DataFrame(FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t['Ge
DataGenre11 = DataGenre11.reset_index()
DataGenre11.rename(columns = {'index':'Genre names', 'Genre':'Frequency'},
df_genre_freq11 = DataGenre11.loc[[True if "Drama" in x else False for x in
list_to_h_c_a_d_r_ab_sff_aa_t_d = [df_genre_freq11["Genre names"][index] fo
 [8,10,11,12,13,14,15,16,18,21,22,23,24,25,26]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_d = FinalCombine_No_NaN_to_h_c
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_d["Genre"] = ["Comedy-Drama" i
DataGenre12 = pd.DataFrame(FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_d['
DataGenre12 = DataGenre12.reset_index()
DataGenre12.rename(columns = {'index':'Genre names', 'Genre':'Frequency'},
df_genre_freq12 = DataGenre12.loc[[True if "Comedy" in x else False for x i
list_to_h_c_a_d_r_ab_sff_aa_t_dc = [df_genre_freq12["Genre names"][index] f
 [6,9]]
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_dc = FinalCombine_No_NaN_to_h_
FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_dc["Genre"] = ["Comedy-Drama"
DataGenre13 = pd.DataFrame(FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_dc[
DataGenre13 = DataGenre13.reset_index()
FinalGenre = FinalCombine_No_NaN_to_h_c_a_d_r_ab_sff_aa_t_dc
FinalGenre.drop([369,454,471,545,522,618,1082,1354], axis=0, inplace=True)
FinalGenrereindex = FinalGenre.reset_index(drop = True)
new_dataset_imdb = FinalGenrereindex
movie_title_imdb = [x.upper()
                    .replace(" : "," ")
                    .replace(": "," ")
                    .replace(" :"," ")
                    .replace(":"," ")
                    .replace(" . "," ")
                    .replace(". "," ")
                    .replace(" ."," ")
                    .replace("."," ")
                    .replace(" - "," ")
                    .replace(" -"," ")
```

```
                                     .replace("- "," ")
                                     .replace("-"," ")
                                     .replace(" \\ ","")
                                     .replace("\\ ","")
                                     .replace(" \\","")
                                     .replace("\\","")
                                     .replace("   "," ")
                                     .replace(" , "," ")
                                     .replace(" ,"," ")
                                     .replace(", "," ")
                                     .replace(","," ")
                                     .replace(" ! "," ")
                                     .replace(" !"," ")
                                     .replace("! "," ")
                                     .replace("!"," ")
                                     .replace(" ? "," ")
                                     .replace(" ?"," ")
                                     .replace("? "," ")
                                     .replace("?"," ")
                                     .replace(" / "," ")
                                     .replace(" /"," ")
                                     .replace("/ "," ")
                                     .replace("/"," ")
                                     .replace(" ( "," ")
                                     .replace("( "," ")
                                     .replace(" ("," ")
                                     .replace("("," ")
                                     .replace(" ) "," ")
                                     .replace(" )"," ")
                                     .replace(") "," ")
                                     .replace(")"," ")
                                     .replace(" [ "," ")
                                     .replace("[ "," ")
                                     .replace(" ["," ")
                                     .replace("["," ")
                                     .replace(" ] "," ")
                                     .replace(" ]"," ")
                                     .replace("] "," ")
                                     .replace("]"," ")
                                     .replace(" * "," ")
                                     .replace(" *"," ")
                                     .replace("* "," ")
                                     .replace("*"," ")
                                     .replace(" & "," AND ")
                                     .replace("& "," AND ")
                                     .replace(" &"," AND ")
                                     .replace("&"," AND ")
                                     .replace("Ä","A")
                                     .replace("É","E")
                                     .replace("·"," ")
                                     .replace("Á","A")
                                   for x in new_dataset_imdb["Title"]]
movie_title_imdb
rare_title_list(movie_title_imdb)
removie_list_imdb = rare_title_list(movie_title_imdb)
#[x for x in movie_title_imdb if x.startswith(" ") or x.endswith(" ")]
movie_title_imdb = [x.rstrip(" ") if x.endswith(" ") else x for x in movie_
```

```
movie_title_imdb = [x.lstrip(" ") if x.startswith(" ") else x for x in movi
clean_movie_title_imbd = pd.DataFrame(movie_title_imdb) #clean movie title
tem_dataset_imdb = pd.concat([new_dataset_imdb,clean_movie_title_imbd],axis
tem_dataset_imdb.columns = [*tem_dataset_imdb.columns[:-1], 'Movie Name']
tem_dataset_imdb = tem_dataset_imdb.loc[[False if x in remobie_list_imdb el
tem_dataset_imdb = tem_dataset_imdb.sort_values(by = 'Movie Name')
tem_dataset_imdb["Title"] = [x + " " + str(y)  for x,y in zip(tem_dataset_i
tem_dataset_imdb = tem_dataset_imdb.reset_index(drop=True)
tem_dataset_imdb = tem_dataset_imdb.iloc[:,0:13]
year_list  = tem_dataset_imdb["Year"]
year_list_new = year_list.apply(lambda st: st[st.find("(")+1:st.find(")")])
tem_dataset_imdb["Year"] = year_list_new
# So our final data is (tem_dataset_imdb)
#tem_dataset_imdb.to_csv("Completed_Dataset_Part2.csv")
part1=pd.read_csv("/Users/lucchen/Desktop/141 final project/Completed_Datas
part2=pd.read_csv("/Users/lucchen/Desktop/141 final project/Completed_Datas
part_1=part1.drop("Unnamed: 0",axis=1) #drop the frist column Unnamed
part_2=part2.drop("Unnamed: 0",axis=1)
part_2.rename({"Title":"Movie"},axis = "columns", inplace = True) #change t
part_1["Movie"]=final_name_change(part_1["Movie"])
part_2["Movie"]=final_name_change(part_2["Movie"])
result_Final=pd.merge(part_1,part_2,on="Movie")
result_final=result_Final.drop("Year",axis=1) #the final merged version dat
df = pd.read_csv("Finalone.csv").iloc[:,1:] #Finalone.csv is result_final d
df["Director"] = [string.replace("[","").replace("]","").replace("'","").sp
df["Year"] = [int(x.split(" (")[1].replace(")","")) for x in df["Movie"]]
url_oscar = 'https://awardsdatabase.oscars.org/search/getresults?query'
query_oscar = {"Sort":"3-Award Category-Chron","AwardShowNumberFrom": 1,"Aw
params_oscar = {'query': json.dumps(query_oscar)} # recasts params to query
headers_oscar = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_1
results_oscar = requests.get(url_oscar, headers = headers_oscar, params = p
results_oscar.raise_for_status()
html_oscar = lx.fromstring(results_oscar.text)
winners = [string.text_content()
           if not "\r" in string.text_content()
           else None
           for string in html_oscar.xpath("//div[@class = 'awards-result-ch
winners = [x for x in winners if x != None]
all_year = []
one_year = []
for x in winners:
    if (x.endswith("st)") or x.endswith("nd)") or x.endswith("rd)") or x.en
        all_year.append(one_year)
        one_year = [x]
    else:
        one_year.append(x)
all_year.append(one_year)
all_year = all_year[1:]
oscar_df = pd.DataFrame(columns = ["Year", "Oscar_Actor", "Oscar_Director"]
for i in range(0,len(all_year)):
    df_year = pd.DataFrame(all_year[i])
    index_actor_title = df_year.loc[[True if x.startswith("ACT") else False
    actor_name = [all_year[i][x+1] for x in index_actor_title]
    actor_name
    index_director_title = df_year.loc[[True if x.startswith("DIRECTING") e
    director_name = [all_year[i][x+1] if i <= 2 else all_year[i][x+2] for x
    director_name
```

```
    year_df = pd.DataFrame([{"Year": all_year[i][0], "Oscar_Actor": actor_n
    oscar_df = pd.concat([oscar_df, year_df], axis = 0)
oscar_df["Year"] = [x for x in range(1928,2021+1)]
df["Oscar_Lead"] = [oscar_indicator_row(df, "Lead", year_of_movie, i) for y
df["Oscar_Director"] = [oscar_indicator_row(df, "Director", year_of_movie,
response_franchise = requests.get("https://www.businessinsider.com/greatest
html_franchise = lx.fromstring(response_franchise.text)
list_franchise = [x.text_content().replace("\xa0","").replace('"',"").repla
list_year = [x.text_content().replace("\xa0","").replace('"',"").replace("
list_year = ["(" + x + ")" for x in list_year]
std_list_franchise = [x.upper()
                      .replace(": "," ")
                      .replace(":"," ")
                      .replace(". "," ")
                      .replace("."," ")
                      .replace(" — "," ")
                      .replace(" - "," ")
                      .replace("-", " ")
                      for x in list_franchise]
std_list_franchise_year = [x + " " + year for x,year in zip(std_list_franch
std_list_franchise_year = [x
                           .replace("HARRY POTTER AND THE DEATHLY HALLOWS P
                           .replace("HARRY POTTER AND THE DEATHLY HALLOWS P
                           .replace("MARVEL'S THE AVENGERS (2012)", "THE AV
                           for x in std_list_franchise_year]
std_list_franchise_year.append("SPIDER MAN INTO THE SPIDER VERSE (2018)")
std_list_franchise_year.append("STAR TREK (2009)")
std_list_franchise_year.append("STAR TREK II THE WRATH OF KHAN (1982)")
std_list_franchise_year.append("STAR TREK INTO DARKNESS (2013)")
std_list_franchise_year.append("STAR TREK V THE FINAL FRONTIER (1989)")
std_list_franchise_year.append("THE BATMAN (2022)")
df["Franchise"] = [1 if x in std_list_franchise_year else 0 for x in df["Mo
df = df.rename(columns = {"Movie":"movie",
                          "Production Budget": "budget",
                          "Domestic Gross": "gross_dom",
                          "Worldwide Gross": "gross_wor",
                          "Genre": "genre",
                          "Movie Rating": "rating",
                          "User Rating": "score_user",
                          "Meta Score": "score_meta",
                          "Duration in mins": "mins",
                          "Director": "director",
                          "Lead": "lead",
                          "User Rating Good/Bad": "score_user_good",
                          "Meta Score Good/Bad": "score_meta_good",
                          "Director Count 0/1": "director_one",
                          "Director Count": "director_num",
                          "Year": "year",
                          "Oscar_Lead": "oscar_lead",
                          "Oscar_Director": "oscar_director",
                          "Franchise": "ip"})
df_final_plus = pd.concat([df["year"],
                           df["movie"],
                           df["gross_dom"],
                           df["gross_wor"],
                           df["budget"],
                           df["score_user"],
```

```
                                    df["score_user_good"],
                                    df["score_meta"],
                                    df["score_meta_good"],
                                    df["rating"],
                                    df["genre"],
                                    df["ip"],
                                    df["lead"],
                                    df["oscar_lead"],
                                    df["director"],
                                    df["director_num"],
                                    df["director_one"],
                                    df["oscar_director"],
                                    df["mins"]], axis = 1)
df_final_plus.to_csv("df_final_plus.csv")
df_final = df_final_plus.drop(columns = ["movie", "director", "lead"])
df_final.to_csv("df_final.csv")
```

In [4]:
```
df_final = pd.read_csv("df_final.csv").iloc[:,1:]
df_final_clean = df_final.copy()
df_final_clean = df_final_clean.loc[[False if x in ["G","Approved","Not Rat
df_final_clean["score_user_good"] = [1 if x >= 7 else 0 for x in df_final_c
df_final_clean["log_year"] = np.log(df_final_clean["year"])
df_final_clean["log_gross_wor"] = np.log(df_final_clean["gross_wor"])
df_final_clean["log_budget"] = np.log(df_final_clean["budget"])
df_final_clean["log_mins"] = np.log(df_final_clean["mins"])
df_final_clean = df_final_clean.drop(columns = ["gross_dom", "director_num"
df_final_clean = df_final_clean.reset_index(drop = True)
df_final_clean_dummies = pd.get_dummies(df_final_clean)
df_final_clean_dummies = df_final_clean_dummies.rename(columns = {"rating_P
                                                                  "rating_P
                                                                  "rating_R
                                                                  "genre_Ac
                                                                  "genre_An
                                                                  "genre_Bi
                                                                  "genre_Co
                                                                  "genre_Co
                                                                  "genre_Dr
                                                                  "genre_Ho
                                                                  "genre_Ro
                                                                  "genre_Sc
```

All of our data comes from the following websites: [IMDB Top 1000 Movies (https://www.imdb.com/search/title/?groups=top_1000&sort=boxoffice_gross_us,desc)](https://www.imdb.com/search/title/?groups=top_1000&sort=boxoffice_gross_us,desc), [IMDB Bottom 1000 Movies (https://www.imdb.com/search/title/?groups=bottom_1000&sort=boxoffice_gross_us,desc)](https://www.imdb.com/search/title/?groups=bottom_1000&sort=boxoffice_gross_us,desc), [The Numbers Budget & Financial Performance (https://www.the-numbers.com/movie/budgets)](https://www.the-numbers.com/movie/budgets), [Oscar Winners (https://awardsdatabase.oscars.org/search/getresults?query=%7B%22Sort%22%3A%223-Award%20Category-Chron%22%2C%22AwardCategory%22%3A%5B%229998%22%2C%221%22%2C%222%22%2](https://awardsdatabase.oscars.org/search/getresults?query=%7B%22Sort%22%3A%223-Award%20Category-Chron%22%2C%22AwardCategory%22%3A%5B%229998%22%2C%221%22%2C%222%22%2) and [Insider Top 27 Movie Franchises (https://www.businessinsider.com/greatest-movie-franchises-all-time-critics-2018-8)](https://www.businessinsider.com/greatest-movie-franchises-all-time-critics-2018-8). We used web scraping to extract the data we wanted from each website and then combined the different datasets into one final dataset. We cleaned our data by deleting any rows that had no values, categorizing several variables, such as genre, and making sure the different datasets matched up well with each other. We used several functions to assist us in creating our final dataset.

Using web scraping, we use a self-defined function to create a data frame that extracts the release date, movie title, production budget, and worldwide gross for 6345 movies from the Numbers website. The data frame contains the information of every movie on the website with modifications needed because of the errors in data entry, such as the NA value or an unknown release date. In the data cleaning process, we use self-defined functions to standardize movie titles by removing movie titles with non-English language characters, non-character symbols, and misspelling and combining similar movie titles from different data frames to expand the information for the movies.

By web scraping from the IMDB websites, we extracted information from the top 1000 movies and bottom 1000 movies, which includes movie titles, meta score, user rating, year, genre, movie rating, duration in minutes, directors, and lead actor. After extracting these 2000 rows of data, we dropped any rows that had no values and ended up with 1663 rows. By looking at our data set, we noticed that there are many different genres for each movie. We simplified the genres by categorizing and simplifying each row into the most dominant genre. In this way, we ended up with 9 different kinds of genres. We also added several new columns with dummy variables, such as director_one, which would indicate whether or not a movie had only one director.

In addition to this, we ultilized an undocumented API when extracting information from the Oscars website. More specifically, we extracted all Oscar winners from 1934 to 2021. We then filtered for winners that were either actors or directors. Afterward, we created functions that would produce new binary variables, which would tell us the movies with an Oscar lead and the movies with an Oscar director.

We also web scraped from the Insider website, extracting movies which are under top franchises selected by movie critics. Some notable franchises include Spider-Man, Batman, Star Wars, and James Bond. After extracting these titles, we created functions that we produce a new indicator variable, which would indicate the movies in our dataset that are part of established franchises.

Data Description

In [5]:
```
display(df_final_clean_dummies)
```

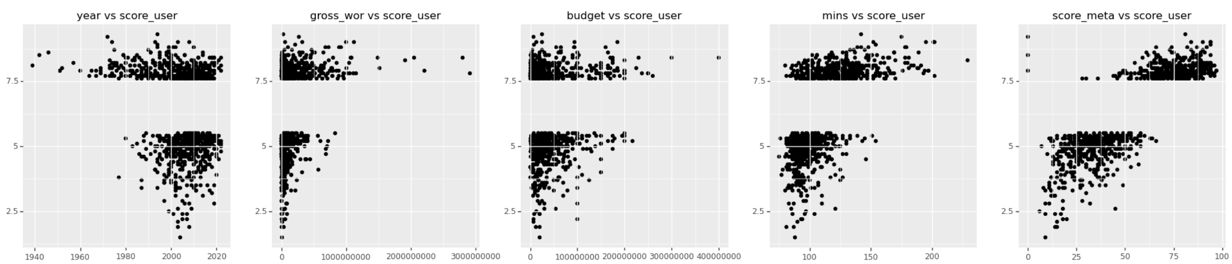
	year	gross_wor	budget	score_user	score_user_good	score_meta	ip	oscar_lead	d
0	2008	2.690657e+08	105000000.0	5.1	0	34.0	0	0	
1	1995	1.688415e+08	29000000.0	8.0	1	74.0	0	0	
2	2013	1.807651e+08	20000000.0	8.1	1	96.0	0	0	
3	2019	3.891404e+08	100000000.0	8.2	1	78.0	0	0	
4	2003	5.966762e+07	20000000.0	7.6	1	70.0	0	0	
...	...	...	...	...	...	...	...	...	
930	2007	8.308008e+07	85000000.0	7.7	1	78.0	0	0	
931	2011	1.708055e+08	80000000.0	5.2	0	30.0	0	0	
932	2016	5.534869e+07	50000000.0	4.7	0	34.0	0	0	
933	2006	1.250619e+07	35000000.0	4.3	0	26.0	0	0	
934	2016	1.004630e+09	150000000.0	8.0	1	78.0	0	0	

935 rows × 27 columns

In the end, our analysis utilizes the dataset shown above, with 935 observations and 27 different variables. These variables include year, gross_wor, budget, score_user, score_user_good, score_meta, ip, oscar_lead, director_one, oscar_director, mins, log_year, log_gross_wor, log_budget, log_mins, rating_pg, rating_pg_13, rating_r, genre_action_adv, genre_animation, genre_bio, genre_comedy, genre_comedy_drama, genre_drama, genre_horror, genre_romance, genre_fantasy_sci. year is the year a movie is released. gross_wor is the world gross of a movie measured in USD. budget is the production budget of movie measured in USD. score_user and score_meta are user ratings, scaled from 0 to 10, and critic ratings, scaled from 0 to 100, with dummy variable score_user_good to indicate whether or not the user rating is greater than or equal to 7. ip is a dummy variable that indicates whether or not a movie is part of an established IP, such as Spider-Man, Batman, Star Wars, and James Bond. oscar_lead and oscar_director are dummy variables that tell us whether or not a movie has an Oscar-winning lead actor and an Oscar-winning director respectively. director_one tells us whether or not a movie has one director. mins is the duration of a movie measured in minutes. log_year, log_gross_wor, log_budget, and log_mins are the log of year, gross_wor, budget, and mins respectively. Each of the rating and genre related variables are indicators that tell us whether or not a movie has a particular rating and whether or not it belongs to a particular genre.

User Score Preliminary Analysis

In [6]:
```
quant_var = ["year", "gross_wor", "budget", "mins", "score_meta"]
fig = (p9.ggplot()+p9.geom_blank(data=df_final_clean_dummies)+p9.theme_void
gs = gridspec.GridSpec(nrows = 1, ncols = 5)
for i,j in zip(quant_var,range(0,5)):
    plot = (
        p9.ggplot(df_final_clean_dummies,p9.aes(x = i, y = "score_user"))
        + p9.geom_point()
    )
    ax = fig.add_subplot(gs[0,j])
    ax.title.set_text(i + " vs score_user")
    plot._draw_using_figure(fig, [ax])
plt.show()
```



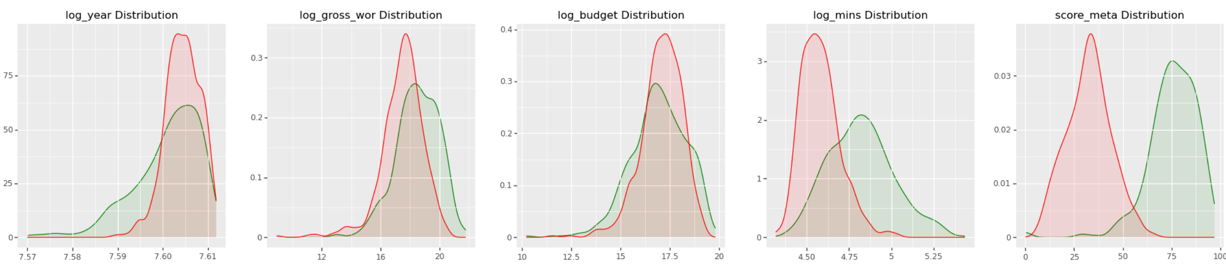
Before applying any methodology, we conducted some premilinary analysis on score_user. We started by creating different scatterplots with score_user as our response variable and year, gross_wor, budget, mins, and score_meta as separate explanatory variables. However, rather than following a linear or a dispersed pattern, points in each plot strictly appear either below or above score_user = 7. This indicates that score_user behaves less like a continuous response and more like a binary response, with one category where score_user is less than 7 or "bad" and another where score_user at least 7 or "good".

In [7]:
```
fig = ff.create_distplot([df_final_clean_dummies["score_user"]], group_labe
fig.update_layout({'plot_bgcolor': 'rgb(235,235,235)'}, showlegend = False,
fig.show()
```

This binary characteristic of score_user is also highlighted in its distribution. As shown above, we can see that the distribution is bimodal, with one center around score_user = 5 and the another around score_user = 8. Again, this inidicates that we can divide score_user into 2 groups, with one group representing bad user scores and another representing "good" user scores.

All of this preliminary analysis is important because it tells us that we can eliminate linear regression as one possible methodology. This is due to the fact that linear regression assumes our response follows a Gaussian distribution; however, as we saw earlier, score_user does not uphold this assumption and instead follows a bimodal distribution. Given the "binary" characteristic revealed by the distribution and scatterplots, it might be better to apply a logististic regression. Unlike linear regression, logistic regression assumes the response of each observation independently follows a Bernoulli distribution. Now, it is quite difficult for us to confirm independence since we lack information on the specific people writing movie reviews (i.e. the score of a horror film will likely not be independent from the score of a romance film if they are both reviewed by people who are hardcore horror fans). Despite this, we can still create a new feature called score_user_good, which will convert score_user into a Bernoulli variable that becomes 1 if score_user is at least 7 or "good" and becomes 0 if score_user is less than 7 or "bad".

In [8]:
```
log_quant_var = ["log_year", "log_gross_wor", "log_budget", "log_mins", "sc
fig = (p9.ggplot()+p9.geom_blank(data=df_final_clean_dummies)+p9.theme_void
gs = gridspec.GridSpec(nrows = 1, ncols = 5)
for i,j in zip(log_quant_var,range(0,5)):
    plot = (
        p9.ggplot(p9.aes(x = i))
        + p9.geom_density(df_final_clean_dummies[df_final_clean_dummies["sc
        + p9.geom_density(df_final_clean_dummies[df_final_clean_dummies["sc
    )
    ax = fig.add_subplot(gs[0,j])
    ax.title.set_text(i + " Distribution")
    plot._draw_using_figure(fig, [ax])
plt.show()
```



With the addition of score_user_good, we can conduct our preliminary analysis with a new perspective. Instead of considering how score_user is correlated with other continuous variables, we can compare different distributions to gain some understanding of how different values for a continuous input affect the probability of a movie getting a good user score.

It is worth mentioning that we also had to create new features such as log_year, log_gross_wor, log_budget, and log_mins which log-transform year, gross_wor, budget, and mins respectively. This is because there were extremely low values for year and extremely high values for gross_wor, budget, and mins, which may cause the effect of other covariates to be diluted. By applying a log-transformation, these variables are put on a common log-scale and won't dilute the impact of any other covariates. We, however, did not apply this to score_meta since certain observations had score_meta = 0 (NOTE: log(0) is undefined). Although, we believe this will not cause too many issues since there are no values for score_meta that are extremely small or large to dilute other covariates.

With that said, the plots above show distributions for log_year, log_gross_wor, log_budget, log_mins, and score_meta given that a movie gets a good user score, which is marked in green, or a bad user score, which is marked in red. For the log_gross_wor and log_budget plots, we can see that there is a lot of overlap between the good and bad user score distributions, which suggests that log_gross_wor and log_budget may each be independent from score_user_good. For the log_year and log_mins plot, there is still a considerable amount of overlap; however, we can see that the probability of a movie getting a good user score is slightly higher when it is recent and slightly lower when it is longer. This suggests that log_year and log_mins may have a slight significant effect on whether or not a movie gets a good user score. For the score_meta plot, there is the least amount of amount overlap, and we can clearly that the probability of a movie getting a good user score given a higher meta score is higher than when given a lower meta score. This suggests that score_meta may have a huge significant effect on whether or not a movie gets a good user score.

In [9]:
```
def conditional_prob_df(list_of_binary_input, df):
    combine_df = pd.DataFrame(index = list_of_binary_input, columns = ["P(s
    for i in list_of_binary_input:
        input_df = pd.crosstab(df["score_user_good"],df[i])
        prob1_given1 = input_df.iloc[1,1]/sum(input_df.iloc[:,1])
        prob1_given0 = input_df.iloc[1,0]/sum(input_df.iloc[:,0])
        combine_df.loc[i,:] = [prob1_given1,prob1_given0]
    diff_prob = abs(combine_df.iloc[:,1] - combine_df.iloc[:,0])
    problem_rows = [x for x,y in zip(list_of_binary_input,diff_prob) if y <
    combine_df_highlight = combine_df.style.applymap(highlight_problem_rows
    display_side_by_side([combine_df_highlight], ["Probability of Good User
conditional_prob_df(['ip',
                     'oscar_lead',
                     'director_one',
                     'oscar_director',
                     'rating_pg',
                     'rating_pg_13',
                     'rating_r',
                     'genre_action_adv',
                     'genre_animation',
                     'genre_bio',
                     'genre_comedy',
                     'genre_comedy_drama',
                     'genre_drama',
                     'genre_horror',
                     'genre_romance',
                     'genre_fantasy_sci'], df_final_clean_dummies)
```

Probability of Good User Score Given X

	P(score_user_good=1|x=1)	P(score_user_good=1|x=0)
ip	0.915254	0.425799
oscar_lead	0.701031	0.428401
director_one	0.452244	0.515152
oscar_director	0.803922	0.436652
rating_pg	0.407821	0.468254
rating_pg_13	0.334262	0.532986
rating_r	0.589421	0.358736
genre_action_adv	0.429204	0.465444
genre_animation	0.568182	0.451178
genre_bio	0.943396	0.427438
genre_comedy	0.169492	0.498164
genre_comedy_drama	0.603175	0.446101
genre_drama	0.746575	0.403042
genre_horror	0.198473	0.498756
genre_romance	0.391304	0.465854
genre_fantasy_sci	0.435897	0.457589

In order to gain some understanding on how our binary covariates may effect score_user_good, we calculated the sample probabilities of getting a good user score when a covariate is either 1 or 0. As we can see in the table above, covariates marked in red are those where these probabilities are roughly equal (NOTE: the difference tolerance that we set is at most 10%). This suggests that a movie getting a good user score may not depend on whether or not it is directed by one director, PG, action/adventure, romance, or sci-fi/fantasy. This leaves us with the unmarked covariates, where the probability of getting a good user score when x = 1 is not equal to when x = 0. This suggests that a movie getting a good user score may depend on whether or not it is part of an established IP, PG-13, R, animation, biography, comedy, comedy-drama, drama, horror, has Oscar leads, or has Oscar directors.

User Score Methodology

In [12]:
```
model_red(df = df_final_clean_dummies, model = "logit", outvar = ["score_us
```

ip	[illegible]	[illegible]	[illegible]	[illegible]
oscar_lead	0.930200	0.281000	3.309000	0.001000
director_one	-0.559000	0.329000	-1.698000	0.089000
oscar_director	1.625100	0.415000	3.917000	0.000000
rating_pg	0.661200	0.263000	2.511000	0.012000
rating_r	1.661300	0.202000	8.244000	0.000000
genre_animation	0.856600	0.424000	2.021000	0.043000
genre_bio	3.425900	0.633000	5.414000	0.000000
genre_comedy	-1.097400	0.319000	-3.439000	0.001000
genre_comedy_drama	0.865200	0.330000	2.620000	0.009000
genre_drama	1.366300	0.273000	4.997000	0.000000
genre_horror	-1.472600	0.315000	-4.671000	0.000000

As of now, we suspect that variables such as log_year, log_mins, score_meta, ip, oscar_lead, oscar_director, rating_pg_13, rating_r, genre_animation, genre_bio, genre_comedy, genre_comedy_drama, genre_drama, and genre_horror may be significant factors that impact whether or not a movie gets a good user score. However, we can put this to the test by first fitting a full logistic model on score_user_good. Although we cannot confirm the independence of each observation, we feel that it is more appropriate to apply logistic regression rather than linear regression because, as shown earlier, score_user does not follow a Gaussian distribution and behaves more like a binary variable.

With that said, the first pair of tables above show the full model summary alongwith VIF measures for each covariate (NOTE: VIF is a measure of how correlated a given covariate is to other covariates, where a value greater than 5 indicates high correlation). As shown in the VIF table, the full model has covariates with extremely large VIF values. This is a problem because this indicates that multicollinearity is present. With multicollinearity, we are more uncertain as to the true effect a particular covariate has on our response, which may explain why some of standard errors are extremely large. One way that we decided to remedy this is by removing covariates with the highest VIF one at a time until all covariates had a VIF value less than 5, which left us with the reduced model summarized above.

User Score Results

Based on our reduced model summary, ip, oscar_lead, oscar_director, rating_pg, rating_r, genre_animation, genre_bio, genre_comedy, genre_comedy_drama, genre_drama, and genre_horror are the significant factors that impact whether or not a movie gets a good user score. Specifically, the estimated difference in log-odds of getting a good user score, holding all other variables constant is 3.44 between movies part of an established IP vs those that aren't, 0.93 between movies with an Oscar lead vs those without one, 1.63 between movies with an Oscar director vs those without one, 0.66 between PG vs non-PG movies, 1.66 between R vs non-R movies, 0.86 between animation vs non-animation movies, 3.43 between biography vs non-biography movies, -1.10 between comedy vs non-comedy movies, 0.87 between comedy-drama vs non-comedy-drama movies, 1.37 between drama vs non-drama movies, and -1.47 between horror vs non-horror movies.

In other words, it is more likely for a movie to get a good user score when it is part of an established IP, PG, R, animation, biography, comedy-drama, drama, Oscar-led, or Oscar-directed. On the other hand, it is less likely for a movie to get a good user score when it is comedy or horror.

However, it is worth noting that some of our coefficient p-values for our reduced model may be "exaggerated" and a lot smaller than what they're supposed to be. As such, the reported effects on the likelihood of getting a good user score may be greater than the actual truth.

User Score Discussion

Although some of our reported effects may be "exaggerated", many of the general trends that we are seeing are quite sensible.

For instance, it is not peculiar to see IP play a significant role in increasing the likelihood of getting a good user score. One possible explanation is that movies part of an established IP like Spider-Man or Batman can bring back warm, nostalgic memories. This may then result in casual viewers feeling overly joyous and sentimental- so much so that they'd feel compelled to write a good review.

In addition to this, it is not surprising that Oscar leads and directors also increase the chance of users giving good scores. This could be explained through the simple notion that these acclaimed individuals possess the best acting and directing skills necessary to provide the most exciting, entralling movie experience. Such captivation is bound to leave audience members more than satisfied to give a good rating.

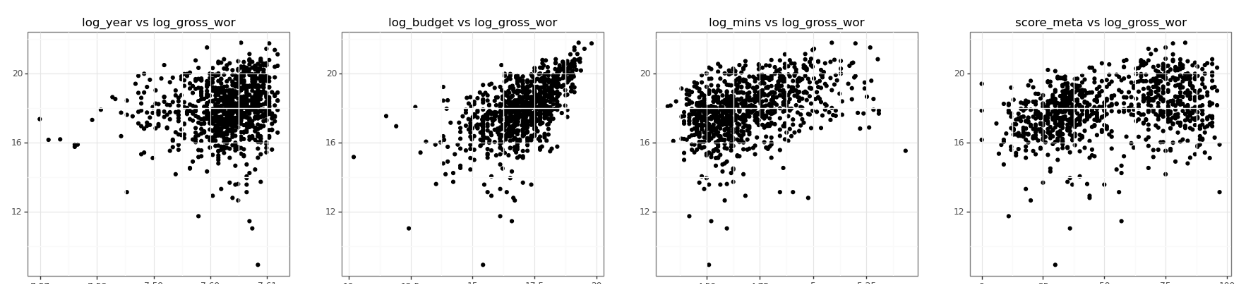
It is also not far-fetched to see genres like biography, drama boost a movie's user score. This is likely connected to how biography and drama are serious, grounded genres, which would push writers to tell deep, compelling stories that engages with movie-goers. This may eventually result in viewers developing a profound connection with a movie and thus giving a good score. A similar explanation could be provided for R movies, but rather than talking about serious genres, we'd be talking about serious ratings.

Animation also appears to boost user score. This could be because animated movies are generally geared towards a younger audience, striving to teach children meaningful life lessons. It is likely that many parents are able to understand these subliminal messages and may feel obligated to post good reviews in order to let other parents know that a particular movie is worth watching with the children. A similar explanation could be provided for PG movies.

While these factors are shown to increase the chance of getting a good user score, comedy and horror movies are shown to decrease this chance. One possible explanation is that comedy and horror movies are generally geared towards teenagers, who often go to the movies to "turn off their brains". This likely results in many producers saving money on "cool fight scenes" rather than talented writers. Without people to write engaging stories, more matured viewers are bound to feel disappointed with their movie experience and leave a poor review.

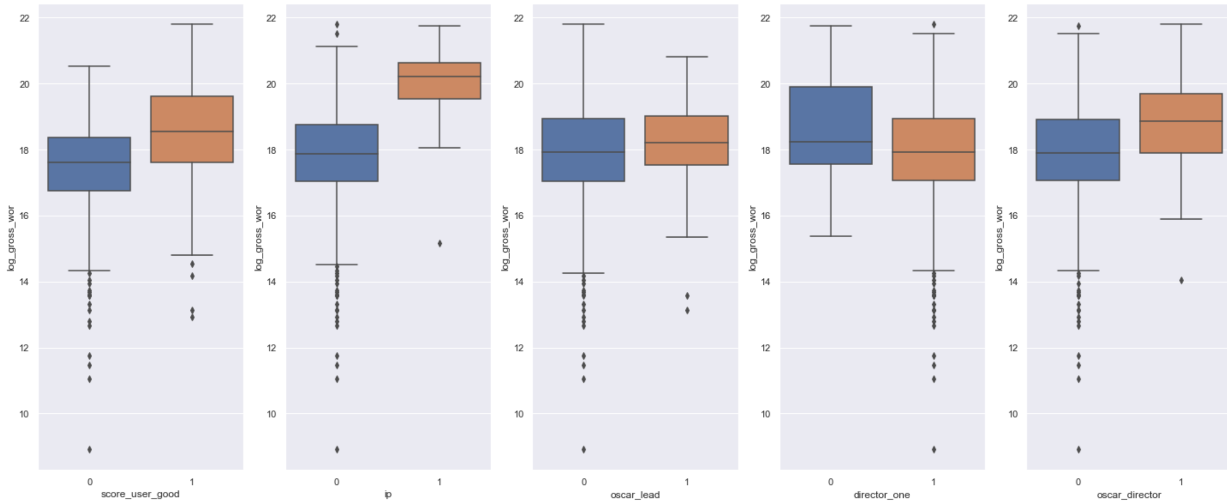
World Gross Preliminary Analysis

In [13]:
```
quant_var = ["log_year", "log_budget", "log_mins", "score_meta"]
fig = (p9.ggplot()+p9.geom_blank(data=df_final_clean_dummies)+p9.theme_void
gs = gridspec.GridSpec(nrows = 1, ncols = 4)
for i,j in zip(quant_var,range(0,4)):
    plot = (
        p9.ggplot(df_final_clean_dummies,p9.aes(x = i, y ="log_gross_wor"))
        + p9.geom_point()
        + p9.theme_bw()
    )
    ax = fig.add_subplot(gs[0,j])
    ax.title.set_text(i + " vs log_gross_wor")
    plot._draw_using_figure(fig, [ax])
plt.show()
```



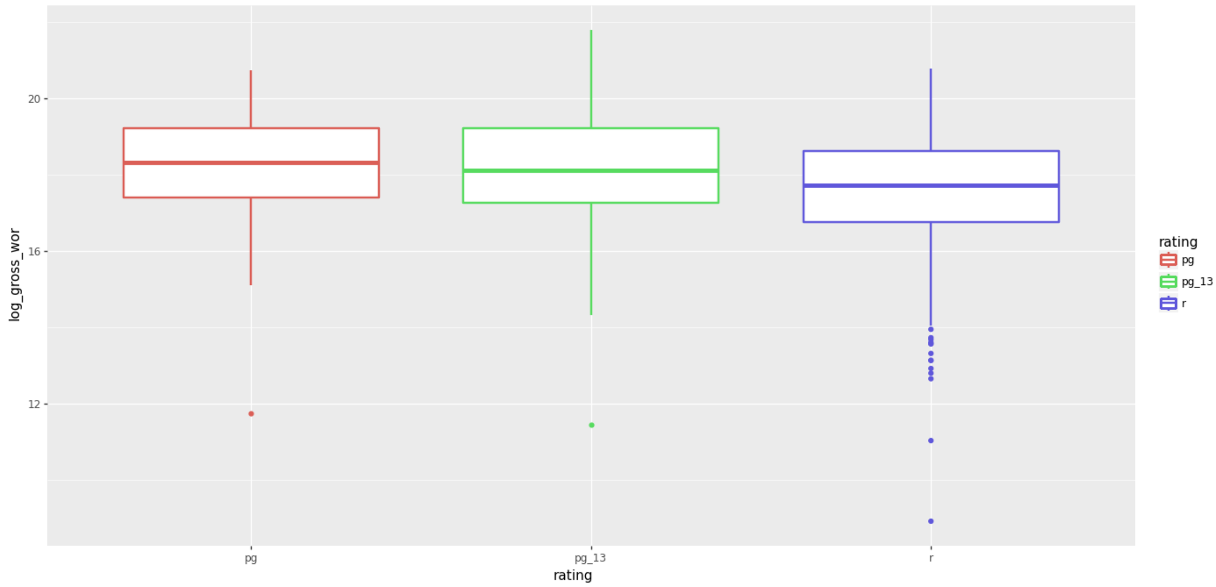
Before applying any methodology, it is worth noting that there is a weak positive correlation or no correlation between the log_gross_wor against other numeric variables in those scatterplots with one exception. The scatterplot for log_budget vs log_gross_wor shows a clear positive correlation between the two variables, unlike the other scatterplots. In context, it suggests that a higher movie budget tends to have a higher gross worldwide, and factors like year, mins, and score meta won’t necessarily increase gross worldwide.

In [14]:
```
sns.set(rc={'figure.figsize':(25,10)})
f, axes = plt.subplots(1, 5)
sns.boxplot(y="log_gross_wor", x= "score_user_good", data=df_final_clean_du
sns.boxplot(y="log_gross_wor", x= "ip", data=df_final_clean_dummies,  orien
sns.boxplot(y="log_gross_wor", x= "oscar_lead", data=df_final_clean_dummies
sns.boxplot(y="log_gross_wor", x= "director_one", data=df_final_clean_dummi
sns.boxplot(y="log_gross_wor", x= "oscar_director", data=df_final_clean_dum
plt.show()
```



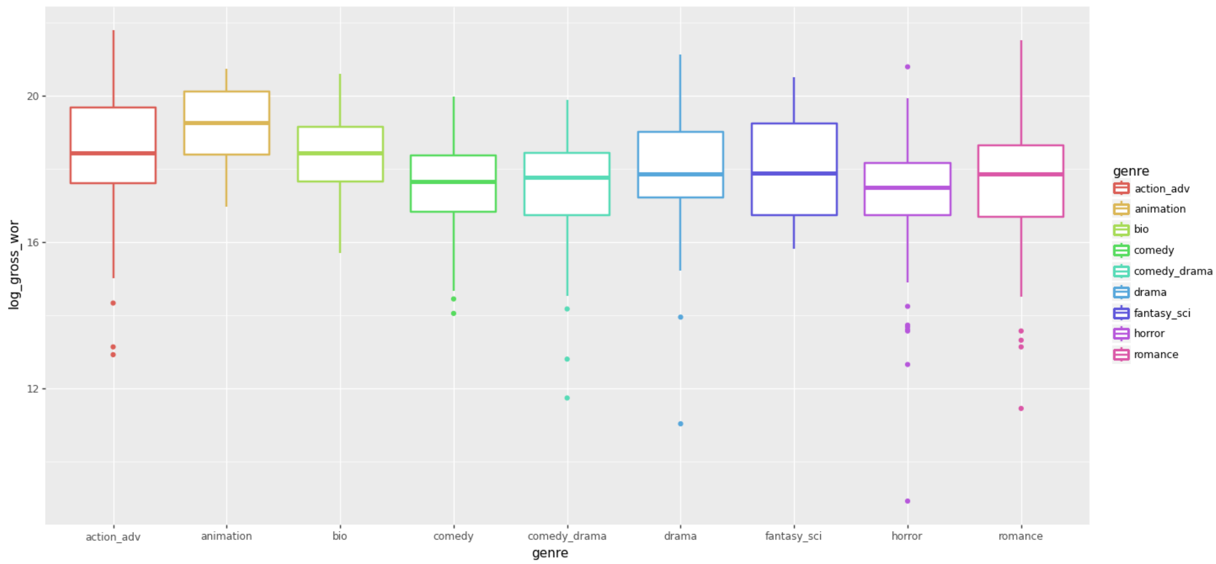
For the boxplots of score_user_good, ip, oscar_lead, director_one, and oscar_director, only the boxplot for oscar_lead and director_one has a similar median and the length of the box overlaps with the other group within the variable, which indicate it's likely that having an oscar lead or having only one director might not make a difference to the movie gross worldwide. For other variables, the median and length of the box are different within each variable, which indicates that having a good user score, an ip movie, or an oscar director tends to make a noticeable difference to the movie gross worldwide.

In [15]:
```
df_rating = df_final_clean_dummies[["log_gross_wor","rating_pg","rating_pg_
df_rating ["rating"] = ["pg" if x==1
                        else "pg_13" if y == 1
                        else "r"
                        for x,y,z in zip(df_rating["rating_pg"],df_rating["
(
    p9.ggplot(df_rating,p9.aes(x="rating",y="log_gross_wor",color="rating")
    + p9.geom_boxplot(size = 1)
    + p9.theme(figure_size=(16, 8))
).draw();
```



For the boxplots of different ratings, it is worth pointing out that all boxplots have roughly the same center and shape, which suggests that ratings might not influence the log_gross_wor.

In [16]:
```
df_genre = df_final_clean_dummies[["log_gross_wor","genre_action_adv","genr
genre_list = [df_genre.iloc[:,i] for i in range(1,10)]
genre_type = ['action_adv','animation','bio','comedy','comedy_drama','drama
genre = pd.DataFrame([None for i in range(len(df_genre))])
for i,j in zip(genre_list,genre_type):
    idx = i.index[i == 1]
    genre.iloc[idx,:] = j
df_genre["genre"] = genre.iloc[:,0]
(
    p9.ggplot(df_genre)  # What data to use
    + p9.aes(x="genre", y="log_gross_wor",color="genre")
    + p9.geom_boxplot(size = 1)
    + p9.theme(figure_size=(16, 8))
).draw();
```



For the boxplots of different genres, it is worth pointing out that all boxplots have roughly the same center and shape, which suggests that genre might not influence the log_gross_wor.

In [17]:
```
fig = ff.create_distplot([df_final_clean_dummies["log_gross_wor"]], group_l
fig.update_layout({'plot_bgcolor': 'rgb(235,235,235)'}, showlegend = False,
fig.show()
```

For the log_gross_wor column, we plotted the distribution of log_gross_wor to check for the normality. Based on the graph, it shows a roughly normal distribution, which suggests we could use linear regression to make inferences between log_gross_wor and other variables. With linear regression, we will need to check additional plot to remove influential points, outliers, and check for error normality and heteroscedasticity.

World Gross Methodology

In [18]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```

The fitted versus residuals plot shows the spread of the residuals is decreasing as the fitted values change, which looks like a funnel shape. The spread tells us the variance is not constant, so we conclude that there is heteroskedasticity. The graph has dashed lines at -3 and 3 with the space shaded in below -3 and above 3, which shows that any point within these intervals is an outlier. From this, we know there are several outliers.

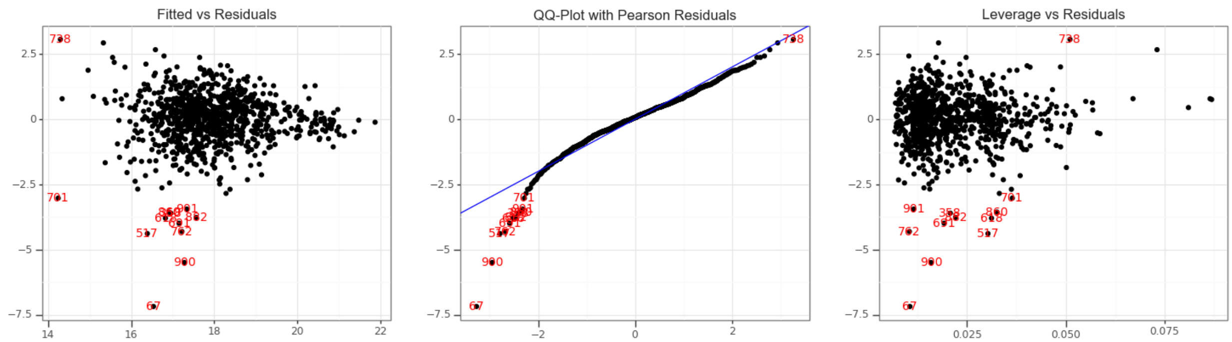
In [19]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```

The second scatter plot is a normality QQ plot, which tells us about the distribution of the residuals. If the errors are normally distributed, we expect the almost all residuals to align with the straight blue line. But in this case, we can see that the distribution is slightly skewed left, meaning that most of the residuals are distributed on the right side with a long tail of residuals extending out to the left, which indicates that the errors may not be normally distributed.

In [20]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```

In the third scatter plot, which is the leverage versus residuals plot, there does not appear to be any influential points. Although there are a considerable amount of outliers, none of them appear to have extremely large leverage. With this information, we know that, at the very least, none of our estimated slopes will be swayed by extreme observations.

In [21]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```



Coefficients

	coef	std err	P>|t|
Intercept	-158.300400	36.108000	0.000000
log_year	30.554700	6.866000	0.000000
log_budget	0.632400	0.039000	0.000000
log_mins	0.657700	0.282000	0.020000
score_meta	0.006800	0.003000	0.018000
score_user_good	0.922500	0.156000	0.000000
ip	0.591900	0.163000	0.000000
oscar_lead	-0.189300	0.118000	0.110000
director_one	-0.309800	0.146000	0.034000
oscar_director	-0.020200	0.159000	0.899000
rating_pg	-52.443400	12.014000	0.000000
rating_pg_13	-52.736600	12.047000	0.000000
rating_r	-53.120400	12.048000	0.000000
genre_action_adv	-17.615400	4.003000	0.000000
genre_animation	-17.731800	4.045000	0.000000
genre_bio	-17.716300	4.015000	0.000000
genre_comedy	-17.499800	4.006000	0.000000
genre_comedy_drama	-17.881700	4.019000	0.000000
genre_drama	-17.594300	4.011000	0.000000
genre_horror	-17.129400	4.011000	0.000000
genre_romance	-17.651800	4.011000	0.000000
genre_fantasy_sci	-17.479900	4.003000	0.000000

Outliers

abs_res

	abs_res
67	7.182360
900	5.495361
517	4.384742
762	4.306547
691	3.986845
618	3.793118
882	3.767409
358	3.601483
860	3.559064
901	3.445995
738	3.066468
701	3.024348

Other Stats

	stat	conclude
error_corr_dw	1.865000	NOT CORRELATED
r_sq_adj	0.520000	MODEL EXPLAIN AT LEAST HALF VAR

The third chart shows that adjusted R squared is 0.52, which means 52% of the variability observed in the target variable is explained by the regression model. We also applied Durbin-Watson test for autocorrelation in the residuals. The range for the Durbin-Watson statistic is from 0 to 4, and we find that our test statistics is 1.87, which indicates there is no autocorrelation of errors.

In [22]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```

After dropping outliers such as observation 67, 900, 517, 762, 691, 618, 882, 358, 860, 901, 701, 260, 659, 496, 738, 830, 718, and 838, the fitted versus residuals plot show the spread of the residuals remaining fairly constant as fitted values change. This suggests that homoscedasticity may be upheld.

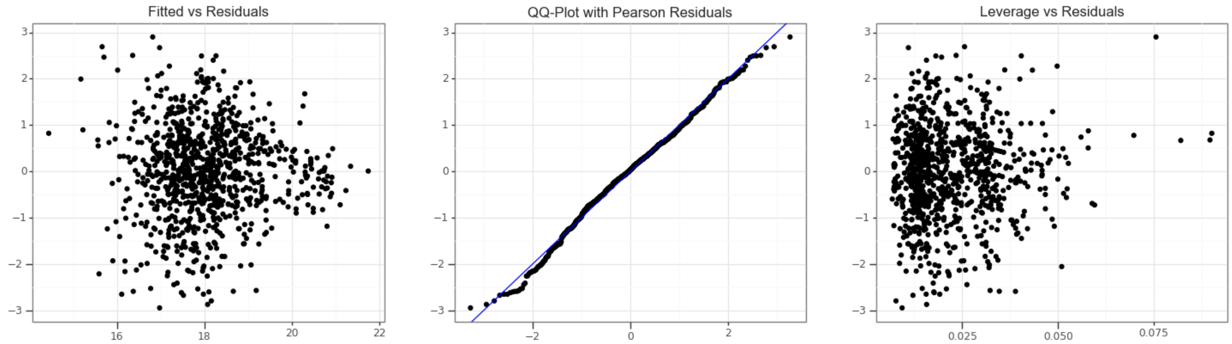
In [23]:
```
_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta + sco
```

After dropping the aforementioned observations, we can see that almost all residuals align with the straight blue line, which indicates that the errors may be normally distributed.

In [24]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```

After dropping the aforementioned observations, there still does not appear to be any influential points. Again, without influential points, we know that, at the very least, none of our estimated slopes will be swayed by extreme observations.

In [25]:
```
lin_assump("log_gross_wor ~ log_year + log_budget + log_mins + score_meta +
```



Coefficients

	coef	std err	P>|t|
Intercept	-154.583900	30.917000	0.000000
log_year	29.827100	5.882000	0.000000
log_budget	0.617800	0.034000	0.000000
log_mins	0.753700	0.244000	0.002000
score_meta	0.006400	0.002000	0.010000
score_user_good	0.816700	0.136000	0.000000
ip	0.628900	0.138000	0.000000
oscar_lead	-0.112100	0.101000	0.269000
director_one	-0.218700	0.124000	0.078000
oscar_director	-0.081900	0.135000	0.544000
rating_pg	-51.203600	10.286000	0.000000
rating_pg_13	-51.559600	10.315000	0.000000
rating_r	-51.820600	10.316000	0.000000
genre_action_adv	-17.208600	3.428000	0.000000
genre_animation	-17.299200	3.463000	0.000000
genre_bio	-17.341300	3.437000	0.000000
genre_comedy	-17.169100	3.431000	0.000000
genre_comedy_drama	-17.391300	3.442000	0.000000
genre_drama	-17.188400	3.434000	0.000000
genre_horror	-16.717000	3.434000	0.000000
genre_romance	-17.129900	3.434000	0.000000
genre_fantasy_sci	-17.139000	3.428000	0.000000

Outliers

abs_res

Other Stats

	stat	conclude
error_corr_dw	1.822000	NOT CORRELATED
r_sq_adj	0.576000	MODEL EXPLAIN AT LEAST HALF VAR

The third chart shows that adjusted R squared is 0.576, which means 57.6% of the variability observed in the target variable is explained by the regression model, which is 5.6% higher than before we dropped the outliers. From the Durbin-Watson statistics, we get a similar result of test statistics which is 1.822 which is also close to 2 and indicates there is no autocorrelation of errors.

In [26]:
```
model_red(f = "log_gross_wor ~ log_year + log_budget + log_mins + score_met
```

Coefficients

	coef	std err	t	P>|t|
Intercept	-154.583900	30.917000	-5.000000	0.000000
log_year	29.827100	5.882000	5.071000	0.000000
log_budget	0.617800	0.034000	18.072000	0.000000
log_mins	0.753700	0.244000	3.092000	0.002000
score_meta	0.006400	0.002000	2.578000	0.010000
score_user_good	0.816700	0.136000	6.016000	0.000000
ip	0.628900	0.138000	4.552000	0.000000
oscar_lead	-0.112100	0.101000	-1.106000	0.269000
director_one	-0.218700	0.124000	-1.763000	0.078000
oscar_director	-0.081900	0.135000	-0.607000	0.544000
rating_pg	-51.203600	10.286000	-4.978000	0.000000
rating_pg_13	-51.559600	10.315000	-4.998000	0.000000
rating_r	-51.820600	10.316000	-5.023000	0.000000
genre_action_adv	-17.208600	3.428000	-5.020000	0.000000
genre_animation	-17.299200	3.463000	-4.995000	0.000000
genre_bio	-17.341300	3.437000	-5.045000	0.000000
genre_comedy	-17.169100	3.431000	-5.004000	0.000000
genre_comedy_drama	-17.391300	3.442000	-5.052000	0.000000
genre_drama	-17.188400	3.434000	-5.005000	0.000000
genre_horror	-16.717000	3.434000	-4.868000	0.000000
genre_romance	-17.129900	3.434000	-4.988000	0.000000
genre_fantasy_sci	-17.139000	3.428000	-5.000000	0.000000

VIF

	VIF
rating_pg_13	inf
rating_r	inf
genre_romance	inf
genre_horror	inf
genre_drama	inf
genre_comedy_drama	inf

	VIF
genre_comedy	inf
genre_bio	inf
genre_animation	inf
genre_action_adv	inf
genre_fantasy_sci	inf
rating_pg	inf
score_user_good	5.235725
score_meta	4.299373
log_mins	2.535535
log_budget	1.924279
log_year	1.420812
ip	1.313645
director_one	1.158891
oscar_director	1.094042
oscar_lead	1.081227

Coefficients of Reduced

	coef	std err	t	P>|t|
Intercept	18.137500	0.182000	99.638000	0.000000
score_user_good	0.860500	0.096000	9.002000	0.000000
ip	1.332500	0.175000	7.617000	0.000000
oscar_lead	0.108700	0.130000	0.834000	0.405000
director_one	-0.108200	0.160000	-0.675000	0.500000
oscar_director	0.360700	0.171000	2.104000	0.036000
rating_pg	-0.078800	0.123000	-0.642000	0.521000
rating_r	-0.551200	0.094000	-5.863000	0.000000
genre_animation	0.604200	0.226000	2.675000	0.008000
genre_bio	-0.318600	0.190000	-1.678000	0.094000
genre_comedy	-0.492900	0.142000	-3.477000	0.001000
genre_comedy_drama	-0.686900	0.174000	-3.943000	0.000000
genre_drama	-0.388400	0.135000	-2.879000	0.004000
genre_horror	-0.352900	0.140000	-2.519000	0.012000
genre_romance	-0.436500	0.140000	-3.121000	0.002000
genre_fantasy_sci	-0.222200	0.205000	-1.085000	0.278000

VIF of Reduced

	VIF
director_one	4.402215
score_user_good	2.798125
rating_r	2.486503
rating_pg	1.951952
genre_drama	1.845546
genre_horror	1.616790
genre_comedy	1.574226
genre_romance	1.480937
genre_animation	1.435119
genre_bio	1.392133
genre_comedy_drama	1.327843
ip	1.298266
oscar_lead	1.184218
genre_fantasy_sci	1.157565
oscar_director	1.112706

Now, we intially fit a full model without dropping any predictor variables; however, even after removing problematic observations, we see that rating_pg_13 has the highest VIF, meaning that rating_pg_13 is highly correlated with at least one other predictor in the model. This is an issue because multicollinearity makes it difficult to determine the true effect a particular covariate has on the response, which would explain why some of our standard errors are considerably large.

One way of remedying this issue is by dropping variables with the largest VIF value. In the end, our final reduced model includes variables such as score_user_good, ip, oscar_lead, director_one, oscar_director, rating_pg, rating_r, genre_animation, genre_bio, genre_comedy, genre_comedy_drama, genre_drama, genre_horror, genre_romance, and genre_fantasy_sci.

World Gross Results

We can say that, holding all other variables constant, the approximate difference in the log of world gross is 0.86 between movies with a good user score vs a bad user score, 1.3 between movies part of an established IP vs those that aren't, 0.36 between movies with an Oscar director vs those without one, -0.55 between R vs non-R movies, -0.60 between animation vs non-animation movies, -0.49 between comedy vs non-comedy movies, -0.69 between comedy & drama vs non-comedy & drama movies, -0.39 between drama vs non-drama movies, -0.35 between horror vs non-horror movies, -0.44 between romance vs non-romance movies.

Simply put, a movie’s world gross tends to be larger when it has a good user score, is part of an established IP, has an Oscar director, and falls under the animation genre. On the other hand, a movie’s world gross tends to be smaller when it is R rated, fall under the comedy, comedy & drama, drama, horror, and romance genres.

However, it is important to note that some of our coefficient p-values for our reduced model may be "exaggerated" and a lot smaller than what they're supposed to be. And so, some of the aforementioned effects on the log of world gross may be greater than the actual truth.

World Gross Discussion

Based on these descriptions, it appears that movies with good user ratings tend to have higher worldwide gross than those that are lower. This is likely connected to the simple notion that a higher rating generally indicates better quality, whether it is because of the plot, genre, or execution of the movie. Potential viewers will look at user ratings to choose which movie to watch, and lower user ratings often turn people away from watching a movie. This then decreases the worldwide gross. Higher user ratings have a similar effect, encouraging people to spend money on a movie.

Movies under a popular franchise IP tend to have a higher worldwide gross than those that aren’t. One possible explanation is that movies with brand recognition are well-known and well-received. These movies already have a large support base. In comparison, standalone movies have to start from the bottom. There is no pre-existing plot nor are there any dedicated fans. Therefore, it makes sense that movies under a popular franchise, such as Star Wars or James Bond, tend to earn more than those that aren’t.

It also appears that movies with an Oscar director earn more than those that don’t. This is likely due to the fact that directors who earn an Oscar know how to direct a captivating movie. Not everyone can earn an Oscar, so those that do are incredibly talented and skilled in their profession. Therefore, we can assume that Oscar directors generally create better movies, which leads to higher viewer traction and higher worldwide gross.

On the other hand, R rated movies seem to generate less worldwide gross, which may be because the rating restricts the audience. From the very start, fewer people can view the movie. Furthermore, even with people who can watch the movie, not everyone would like to spend more money in order to watch it again.

While R rated movies generate less income because of the restricted audience, animated movies seem to generate more worldwide gross compared to all the other types of movie genres. Movies that are under the animation genre are generally family-friendly, which means anyone can watch it. Furthermore, animation is often well-received from most people, possibly because the style is more appealing compared to movies with real life actors.

Meanwhile, movies that fall under the comedy, comedy-drama, drama, horror, and romance genres all generate less worldwide gross, likely because movies under these genres do not attract as many viewers. Genres like comedy, comedy-drama, and drama are similar to each other in that they all have a certain type of exaggeration involved in the plot, whether for conflict or laughs. In addition to this, horror is not for everyone, since it involves gore, jump scares, eerie music, and similar elements. Lastly, not everyone would like the overly sentimental plotlines in romance movies. These factors all decrease the number of potential viewers, and therefore the worldwide gross for these movies.

Conclusion

In the end, we found through logistic regression techniques there were 11 significant factors that impact whether or not a movie gets a good user score. These include ip, oscar_lead, oscar_director, rating_pg, rating_r, genre_animation, genre_bio, genre_comedy, genre_comedy_drama, genre_drama, and genre_horror. There are 9 variables that tend to increase the chance of a movie getting a good user score, which are IP, a PG rating, R rating, animation, biography, comedy-drama, drama, Oscar led, or Oscar directed. There are 2 variables that tend to decrease the chance of a movie getting a good user score which are comedy or horror.

Furthermore, we found through linear regression techniques that movies that are under a popular franchise, have an Oscar director, or are animated have a higher worldwide gross than those that are not. We found that animated movies generally earn higher worldwide gross than any other type of genre, which are comedy, comedy-drama, drama, horror, and romance. Meanwhile, R rated movies tend to have lower worldwide gross compared to a PG or PG-13 rated movie.

With that said, it is worth noting there were several technical difficulties that we faced throughout our analysis.

Before fully cleaning the dataset, there was only one genre column, the majority of which had multiple different genres to describe each movie. To make our data analysis more concise, we had to go through each movie and its genre one by one. For example, we first looked at all the indices that had horror as part of its listed genres, and then created a new list of genres that simplifies it to only include movies that we determined were only horror. A movie could have horror, action, and thriller as its genres, and then we would simplify it down to just horror. This process was very subjective, since we had to decide whether each movie fit into one of the genres more than the other listed ones. We also decided what those groups of genres were. Because of its inherent subjectiveness, we could have movies that other people would not see as our chosen genre.

After standardization, we thought that we could merge both our IMDB and Numbers data perfectly, but we found that there were some movie names that were different, for example, there would be two rows right next to each other with the same movie title instead of one row because of a mismatch in the title string, such as "10 000 B C (2008)" and "10 000 BC (2008)" so we decided to look up each row and replace the movie name consistently so that we can merge by the correct movie title. Also, there are some movie years that are different such as “TEETH (2008)" and "TEETH (2007)”, so we need to replace them with the correct movie year manually.

On the Oscars website, most events were shown to be held during one year; however, there were some that were shown to be held during 1934/1935. In order to remedy this issue, we decided to select the first year shown with events held during 2 years. By doing this, each event would correspond to one, unique year, which we believed was appropriate since Oscars are only held once a year. Despite our best efforts, we believe that slight discrepancies may still have been present in our final data. This is because other websites would claim that the 2000 Oscars were held in 2001. Unfortunately, we couldn’t determine why these inconsistencies were occurring among different sources. In the end, we believed it was best to use the years listed on the Oscars website since the people managing the site are part of the same organization that holds the Oscar events.

We also extracted information from the Insider website, which listed the top 27 franchises according to movie critics. One inherent issue that arises when extracting this type of information is determining what sources are defining as “top”. As mentioned earlier, this list was composed by movie critics; however, the site doesn't make it explicitly clear what metrics critics used. The site would mention that franchises such as the MCU and Star Wars have dominated the box office while others have would earn more critical acclaim; however, we couldn’t determine if variables such as world gross and movie score were variables that critics specifically considered.

Sources

IMDB Top 1000 Movies (https://www.imdb.com/search/title/?groups=top_1000&sort=boxoffice_gross_us,desc)

IMDB Bottom 1000 Movies (https://www.imdb.com/search/title/?groups=bottom_1000&sort=boxoffice_gross_us,desc)

The Numbers Budget & Financial Performance (https://www.the-numbers.com/movie/budgets)

Oscar Winners (https://awardsdatabase.oscars.org/search/getresults?query=%7B%22Sort%22%3A%223-Award%20Category-Chron%22%2C%22AwardCategory%22%3A%5B%229998%22%2C%221%22%2C%222%22%2

Insider Top 27 Movie Franchises (https://www.businessinsider.com/greatest-movie-franchises-all-time-critics-2018-8)