In [424…

```
import re
import statistics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
```

In [425…

```
# load the dataset, and skip the header/first row
data = np.genfromtxt('movieReplicationSet.csv', delimiter = ',', skip_header = 1
# read the csv file as a dataframe in pandas
df = pd.read_csv('movieReplicationSet.csv')
# get a subset of the dataset for just the movie ratings
movies = df.iloc[:,0:400]
# setting per-test significance level α to 0.005
alpha = 0.005
# df.shape - 1097x477
# df[:0] # row 0 - header row
# df.
# df['The Life of David Gale (2003)'] # column for the first movie
# df.iloc[:,0] # column for the first movie
# df.iloc[:,0:2] - the first two columns
# round(np.mean(data[:,0][np.isfinite(data[:,0])]),5) == round(np.mean(df['The L
# sum(df.iloc[:,0].isna()) - number of n/a in the first column
# movies.isna().sum() - n/a in each column
movies.isna().sum() > statistics.median(movies.isna().sum())
```

Out[425…

```
The Life of David Gale (2003)                 True
Wing Commander (1999)                         True
Django Unchained (2012)                      False
Alien (1979)                                 False
Indiana Jones and the Last Crusade (1989)    False
                                             ...
Patton (1970)                                 True
Anaconda (1997)                              False
Twister (1996)                                True
MacArthur (1977)                              True
Look Who's Talking (1989)                     True
Length: 400, dtype: bool
```

In [150…

```
# Question 1:
# Are movies that are more popular (operationalized as having more ratings) rate
# movies that are less popular?

# first off, find the median for the popularity of movies
med = statistics.median(movies.isna().sum()) # 899.5

# and then, do a median-split of popularity to determine high vs. low popularity
highpopularitymovies = movies.loc[:,movies.isna().sum() < med]
lowpopularitymovies = movies.loc[:,movies.isna().sum() > med]

# replacing missing values with column means
highpopularitymovies_nmv = highpopularitymovies.fillna(highpopularitymovies.mean
lowpopularitymovies_nmv = lowpopularitymovies.fillna(lowpopularitymovies.mean())

# Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
# Test for comparing medians of ordinal data (such as movie ratings) from 2 grou
```

```
u1,p1 = stats.mannwhitneyu(highpopularitymovies_nmv,lowpopularitymovies_nmv) # (
print(p1 < alpha) # reject null hypothesis
```

```
True
```

In [394…

```
# Question 2:
# Are movies that are newer rated differently than movies that are older?

# first off, extract the release year for each movie / column
movies_year = movies.copy()
movie_names = []
for i in range(400):
    movie_names.append(movies.columns[i])
# adding a new row in the end to store the movie names as strings
movies_year.loc[len(movies_year.index)] = movie_names
# extract the release year using regex
years = movies_year.iloc[-1].str.extract(r'^.*?\([^\d]*(\d+)[^\d]*\).*$')
# converting the years extracted from a dataframe to a Series
years = pd.Series(years[0])
# then changing the datatype of the years from strings to numeric
years = pd.to_numeric(years)
# adding a new row in the end to store only the years
movies_year.loc[len(movies_year.index)] = years
# taking care of the only missing data by filling it in manually
movies_year['Rambo: First Blood Part II'][1098] = 1985

# secondly, find the median of movies by year of release
med2 = statistics.median(movies_year.iloc[1098]) # 1999

# doing a median split of year of release for old and new movies
oldmovies = movies_year.loc[:,movies_year.iloc[-1] < med2] # 197 movies
movies1999 = movies_year.loc[:,movies_year.iloc[-1] == med2] # 29 movies
newmovies = movies_year.loc[:,movies_year.iloc[-1] > med2] # 174 movies

# manually assign movies that are released in 1999 which is the median
# to new and old movie datasets in order for the two sets have equal length
old1999 = movies1999.iloc[:,:3]
new1999 = movies1999.iloc[:,3:]
oldmovies = pd.concat([old1999, oldmovies], axis=1)
newmovies = pd.concat([new1999, newmovies], axis=1)

# drop the last two columns before computations and hypothesis testing
oldmovies = oldmovies.drop([1097, 1098])
newmovies = newmovies.drop([1097, 1098])

# replacing missing values with column means
newmovies_nmv = newmovies.fillna(newmovies.mean())
oldmovies_nmv = oldmovies.fillna(oldmovies.mean())

# Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
# Test for comparing medians of ordinal data (such as movie ratings) from 2 grou
u2,p2 = stats.mannwhitneyu(newmovies_nmv,oldmovies_nmv) # (-239078306.0, 0.0)
print(p2 < alpha) # reject null hypothesis
```

```
True
```

In [557…

```
# Question 3:
# Is enjoyment of 'Shrek (2001)' gendered, i.e. do male and female viewers rate

# getting a new dataset by making a copy of the original one
```

```
movies_gender = df.copy()
# extracting column of gender and shrek ratings
gender = movies_gender.iloc[:,474]
shrek = movies_gender['Shrek (2001)']

# concatenating movie ratings of shrek with gender identification
shrek_gender = pd.concat([shrek, gender], axis=1)
# getting rid of missing gender entry row-wise
shrek_gender_nmg = shrek_gender.copy()
shrek_gender_nmg = shrek_gender_nmg.dropna(how='any', subset=['Gender identity (

# dividing the dataset into three subsets by viewers' gender identification
shrek_female = shrek_gender_nmg[shrek_gender_nmg.iloc[:,1].astype(int) == 1]
shrek_male = shrek_gender_nmg[shrek_gender_nmg.iloc[:,1].astype(int) == 2]
shrek_sd = shrek_gender_nmg[shrek_gender_nmg.iloc[:,1].astype(int) == 3]

# getting rid of missing movie ratings element-wise
shrek_female = shrek_female.iloc[:,0].dropna()
shrek_male = shrek_male.iloc[:,0].dropna()
shrek_sd = shrek_sd.iloc[:,0].dropna()

# Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
# Test for comparing medians of ordinal data (such as movie ratings) from 2 grou
u3,p3 = stats.mannwhitneyu(shrek_female,shrek_male) # (82232.5, 0.02526831296277
print(p3 < alpha) # do not reject null hypothesis

# Kruskal Wallis test for a non-parametric test for more than two groups
h,p = stats.kruskal(shrek_female,shrek_male,shrek_sd) # (3.8511851030469844, 0.1
print(p < alpha) # do not reject null hypothesis
```

```
False
False
```

In [558…

```
# Question 4:
# What proportion of movies are rated differently by male and female viewers?

# first of all, getting a new dataset by making a copy of the original one
moviegender = df.copy()
# extracting the gender column
gender = moviegender.iloc[:,474]
# create a function that determines whether a single movie is rated
# differently by male and female viewers
def moviebygender(movie_num): # 0 - 399
    # extracting column of movie ratings
    rating = moviegender.iloc[:,movie_num]
    # concatenating movie ratings of shrek with gender identification
    movieandgender = pd.concat([rating, gender], axis=1)
    # getting rid of missing gender entry row-wise
    movieandgender_nmg = movieandgender.copy()
    movieandgender_nmg = movieandgender_nmg.dropna(how='any', subset=['Gender id
    # dividing the dataset into two subsets by viewers' gender identification
    movie_female = movieandgender_nmg[movieandgender_nmg.iloc[:,1].astype(int) =
    movie_male = movieandgender_nmg[movieandgender_nmg.iloc[:,1].astype(int) ==
    # getting rid of missing movie ratings element-wise
    movie_female = movie_female.iloc[:,0].dropna()
    movie_male = movie_male.iloc[:,0].dropna()
    # Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
    # Test for comparing medians of ordinal data (such as movie ratings) from 2
    u0,p0 = stats.mannwhitneyu(movie_female, movie_male)
    return(p0 < alpha) # whether to reject null hypothesis
```

```
# iterate over 400 movies to find the ratio
ratio = 0
for i in range(400):
    yn = moviebygender(i)
    if yn == True:
        ratio += 1
ratio = ratio/400
print(ratio) # 0.1525
```

```
0.1525
```

In [550…

```
# Question 5:
# Do people who are only children enjoy 'The Lion King (1994)' more than people

# first of all, getting a new dataset by making a copy of the original one
movie_onlychild = df.copy()
# extracting the ratings for The Lion King and the only child column
onlychild = movie_onlychild.iloc[:,475]
lionking = movie_onlychild['The Lion King (1994)']
# concatenating movie ratings for The Lion King with the only child column
lionking_onlychild = pd.concat([lionking, onlychild], axis=1)

# getting rid of missing only child entry row-wise
lionking_onlychild_nmoc = lionking_onlychild.copy()
lionking_onlychild_nmoc = lionking_onlychild_nmoc.drop(lionking_onlychild_nmoc.i
# dividing the dataset into two subsets by whether the viewer's a only child or
lionking_onlychild_oc = lionking_onlychild_nmoc[lionking_onlychild_nmoc.iloc[:,1
lionking_onlychild_sl = lionking_onlychild_nmoc[lionking_onlychild_nmoc.iloc[:,1

# getting rid of missing movie ratings element-wise
lionking_onlychild_oc = lionking_onlychild_oc.iloc[:,0].dropna()
lionking_onlychild_sl = lionking_onlychild_sl.iloc[:,0].dropna()

"""
# alternative way of doing this --
# getting rid of missing movie ratings by filling with mean rating
lionking_onlychild_oc = lionking_onlychild_oc.fillna(lionking_onlychild_oc.iloc[
lionking_onlychild_sl = lionking_onlychild_sl.fillna(lionking_onlychild_sl.iloc[
# this way, u and p value we get are (69637.0, 0.0041145679057361575)
# therefore, we reject the null hypothesis
"""
# Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
# Test for comparing medians of ordinal data (such as movie ratings) from 2 grou
u4,p4 = stats.mannwhitneyu(lionking_onlychild_oc,lionking_onlychild_sl) # (52929
print(p4 < alpha) # do not reject null hypothesis
```

```
False
```

In [554…

```
# Question 6:
# What proportion of movies exhibit an "only child effect", i.e. are rated diffe
# by viewers with siblings vs. those without?

# first of all, getting a new dataset by making a copy of the original one
movieonlychild = df.copy()
# extracting the only child column
onlychild = movieonlychild.iloc[:,475]
# create a function that determines whether a single movie is rated
# differently by only children or viewers with siblings
def moviebyonlychild(movie_num): # 0 - 399
```

```
    # extracting column of movie ratings
    rating = movieonlychild.iloc[:,movie_num]
    # concatenating movie ratings with only child identification
    movie_onlychild = pd.concat([rating, onlychild], axis=1)
    # getting rid of missing only child entry row-wise
    movie_onlychild_nmoc = movie_onlychild.copy()
    movie_onlychild_nmoc = movie_onlychild_nmoc.drop(movie_onlychild_nmoc.index[
    # dividing the dataset into two subsets by whether viewers have siblings
    movie_onlychild_oc = movie_onlychild_nmoc[movie_onlychild_nmoc.iloc[:,1].ast
    movie_onlychild_sl = movie_onlychild_nmoc[movie_onlychild_nmoc.iloc[:,1].ast

    # getting rid of missing movie ratings element-wise
    movie_onlychild_oc = movie_onlychild_oc.iloc[:,0].dropna()
    movie_onlychild_sl = movie_onlychild_sl.iloc[:,0].dropna()
    """
    # alternative way of doing this --
    # getting rid of missing movie ratings by filling with mean rating
    movie_onlychild_oc = movie_onlychild_oc.fillna(movie_onlychild_oc.iloc[:,0].
    movie_onlychild_sl = movie_onlychild_sl.fillna(movie_onlychild_sl.iloc[:,0].
    # the ratio we get would be 0.9625 if doing it this way
    """
    # Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
    # Test for comparing medians of ordinal data (such as movie ratings) from 2
    u0,p0 = stats.mannwhitneyu(movie_onlychild_oc,movie_onlychild_sl)
    return(p0 < alpha) # whether to reject null hypothesis

# iterate over 400 movies to find the ratio
ratio2 = 0
for i in range(400):
    yn = moviebyonlychild(i)
    if yn == True:
        ratio2 += 1
ratio2 = ratio2/400
print(ratio2) # 0.035
```

```
0.035
```

In [566…

```
# Question 7:
# Do people who like to watch movies socially enjoy 'The Wolf of Wall Street (20
# more than those who prefer to watch them alone?

# first of all, getting a new dataset by making a copy of the original one
movie_social = df.copy()
# extracting the ratings for The Wolf of Wall Street and the social column
social = movie_social.iloc[:,476]
WolfofWallStreet = movie_social['The Wolf of Wall Street (2013)']
# concatenating movie ratings for The Wolf of Wall Street with the social column
WolfofWallStreet_social = pd.concat([WolfofWallStreet, social], axis=1)

# getting rid of missing social entry row-wise
WolfofWallStreet_social_nms = WolfofWallStreet_social.copy()
WolfofWallStreet_social_nms = WolfofWallStreet_social_nms.drop(WolfofWallStreet_
# dividing the dataset into two subsets by whether the viewer's a only child or
WolfofWallStreet_social_alone = WolfofWallStreet_social_nms[WolfofWallStreet_soc
WolfofWallStreet_social_social = WolfofWallStreet_social_nms[WolfofWallStreet_so

# getting rid of missing movie ratings element-wise
WolfofWallStreet_social_alone = WolfofWallStreet_social_alone.iloc[:,0].dropna()
WolfofWallStreet_social_social = WolfofWallStreet_social_social.iloc[:,0].dropna
```

```
# Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
# Test for comparing medians of ordinal data (such as movie ratings) from 2 grou
u5,p5 = stats.mannwhitneyu(WolfofWallStreet_social_alone,WolfofWallStreet_social
print(p5 < alpha) # do not reject null hypothesis
u5,p5
```

```
False
```

Out[566… `(49303.5, 0.05638214666114465)`

In [578…

```
# Question 8:
# What proportion of movies exhibit such a “social watching” effect?

# first of all, getting a new dataset by making a copy of the original one
moviesocial = df.copy()
# extracting the social column
social = moviesocial.iloc[:,476]
# create a function that determines whether a single movie is rated
# differently by viewers who enjoy watching movies alone or with others
def moviebysocial(movie_num): # 0 - 399
    # extracting column of movie ratings
    rating = moviesocial.iloc[:,movie_num]
    # concatenating movie ratings with social identification
    movie_social = pd.concat([rating, social], axis=1)
    # getting rid of missing social entry row-wise
    movie_social_nmg = movie_social.copy()
    movie_social_nmg = movie_social_nmg.drop(movie_social_nmg.index[movie_social
    # dividing the dataset into two subsets by viewers' social identification
    movie_social_alone = movie_social_nmg[movie_social_nmg.iloc[:,1].astype(int)
    movie_social_social = movie_social_nmg[movie_social_nmg.iloc[:,1].astype(int

    # getting rid of missing movie ratings element-wise
    movie_social_alone = movie_social_alone.iloc[:,0].dropna()
    movie_social_social = movie_social_social.iloc[:,0].dropna()

    # Nonparametric tests equivalent to t-tests - Mann-Whitney U test:
    # Test for comparing medians of ordinal data (such as movie ratings) from 2
    u0,p0 = stats.mannwhitneyu(movie_social_alone, movie_social_social)
    return(p0 < alpha) # whether to reject null hypothesis

# iterate over 400 movies to find the ratio
ratio3 = 0
for i in range(400):
    yn = moviebysocial(i)
    if yn == True:
        ratio3 += 1
ratio3 = ratio3/400
print(ratio3) # 0.0325
```

```
0.0325
```

In [588…

```
# Question 9:
# Is the ratings distribution of ‘Home Alone (1990)’
# different than that of ‘Finding Nemo (2003)’?

# first of all, getting a new dataset by making a copy of the original one
movie_dist = df.copy()
# extracting the ratings for The Wolf of Wall Street and the social column
homealone = movie_dist['Home Alone (1990)']
findingnemo = movie_dist['Finding Nemo (2003)']
```

```
# getting rid of missing movie ratings by removing them element-wise
homealone = homealone.dropna()
findingnemo = findingnemo.dropna()

# doing a 2 sample Kolmogorov-Smirnov test to compare the two movies' distributi
s6,p6 = stats.ks_2samp(homealone, findingnemo) #(0.15269080020897632, 6.37938146
print(p6<alpha) # reject null hypothesis
```

```
True
```

In [615…

```
# Question 10:
# There are ratings on movies from several franchises (['Star Wars', 'Harry Pott
# 'The Matrix', 'Indiana Jones', 'Jurassic Park', 'Pirates of the Caribbean', 'T
# 'Batman']) in this dataset. How many of these are of inconsistent quality,
# as experienced by viewers? [Hint: You can use the keywords in quotation marks
# in this question to identify the movies that are part of each franchise]

# first of all, getting a new dataset by making a copy of the original one
movie_fran = df.copy()
# extracting the ratings for the 8 franchises listed and get rid of missing movi
StarWars0 = movie_fran['Star Wars: Episode IV - A New Hope (1977)'].dropna()
StarWars1 = movie_fran['Star Wars: Episode V - The Empire Strikes Back (1980)'].
StarWars2 = movie_fran['Star Wars: Episode VI - The Return of the Jedi (1983)'].
StarWars3 = movie_fran['Star Wars: Episode 1 - The Phantom Menace (1999)'].dropn
StarWars4 = movie_fran['Star Wars: Episode II - Attack of the Clones (2002)'].dr
StarWars5 = movie_fran['Star Wars: Episode VII - The Force Awakens (2015)'].drop
HarryPotter0 = movie_fran["""Harry Potter and the Sorcerer's Stone (2001)"""].dr
HarryPotter1 = movie_fran['Harry Potter and the Chamber of Secrets (2002)'].drop
HarryPotter2 = movie_fran['Harry Potter and the Goblet of Fire (2005)'].dropna()
HarryPotter3 = movie_fran['Harry Potter and the Deathly Hallows: Part 2 (2011)']
TheMatrix0 = movie_fran['The Matrix (1999)'].dropna()
TheMatrix1 = movie_fran['The Matrix Revolutions (2003)'].dropna()
TheMatrix2 = movie_fran['The Matrix Reloaded (2003)'].dropna()
IndianaJones0 = movie_fran['Indiana Jones and the Raiders of the Lost Ark (1981)
IndianaJones1 = movie_fran['Indiana Jones and the Temple of Doom (1984)'].dropna
IndianaJones2 = movie_fran['Indiana Jones and the Last Crusade (1989)'].dropna()
IndianaJones3 = movie_fran['Indiana Jones and the Kingdom of the Crystal Skull (
JurassicPark0 = movie_fran['Jurassic Park (1993)'].dropna()
JurassicPark1 = movie_fran['The Lost World: Jurassic Park (1997)'].dropna()
JurassicPark2 = movie_fran['Jurassic Park III (2001)'].dropna()
PiratesoftheCaribbean0 = movie_fran['Pirates of the Caribbean: The Curse of the
PiratesoftheCaribbean1 = movie_fran["""Pirates of the Caribbean: Dead Man's Ches
PiratesoftheCaribbean2 = movie_fran["""Pirates of the Caribbean: At World's End
ToyStory0 = movie_fran['Toy Story (1995)'].dropna()
ToyStory1 = movie_fran['Toy Story 2 (1999)'].dropna()
ToyStory2 = movie_fran['Toy Story 3 (2010)'].dropna()
Batman0 = movie_fran['Batman (1989)'].dropna()
Batman1 = movie_fran['Batman & Robin (1997)'].dropna()
Batman2 = movie_fran['Batman: The Dark Knight (2008)'].dropna()

# Kruskal Wallis test for a non-parametric test for more than two groups
h7,p7 = stats.kruskal(StarWars0,StarWars1,StarWars2,StarWars3,StarWars4,StarWars
print(p7 < alpha) # reject null hypothesis
h8,p8 = stats.kruskal(HarryPotter0,HarryPotter1,HarryPotter2,HarryPotter3) # (3.
print(p8 < alpha) # do not reject null hypothesis
h9,p9 = stats.kruskal(TheMatrix0,TheMatrix1,TheMatrix2) # (48.37886652130624, 3.
print(p9 < alpha) # reject null hypothesis
h10,p10 = stats.kruskal(IndianaJones0,IndianaJones1,IndianaJones2,IndianaJones3)
print(p10 < alpha) # reject null hypothesis
```

```
h11,p11 = stats.kruskal(JurassicPark0,JurassicPark1,JurassicPark2) # (46.5908806
print(p11 < alpha) # reject null hypothesis
h12,p12 = stats.kruskal(PiratesoftheCaribbean0,PiratesoftheCaribbean1,Piratesoft
print(p12 < alpha) # reject null hypothesis
h13,p13 = stats.kruskal(ToyStory0,ToyStory1,ToyStory2) # (24.385994936261316, 5.
print(p13 < alpha) # reject null hypothesis
h14,p14 = stats.kruskal(Batman0,Batman1,Batman2) # (190.53496872634642, 4.225296
print(p14 < alpha) # reject null hypothesis
h14,p14
```

```
True
False
True
True
True
True
True
True
```

Out[615… (190.53496872634642, 4.2252969509030006e-42)

In [627…

```
# Extra Credit:
# I would like to explore whether a viewer's tendency to cry during a movie woul
# have an impact on ratings for 'Titanic (1997)'

# first of all, getting a new dataset by making a copy of the original one
df2 = df.copy()
# extracting the ratings for Titanic (1997) and the tendency to cry column
Titanic = df2['Titanic (1997)']
cry = df2['I have cried during a movie']
# concatenating movie ratings for Titanic (1997) with the tendency to cry column
Titanic_cry = pd.concat([Titanic, cry], axis=1)

# getting rid of missing tendency to cry entry row-wise
Titanic_cry_nmc = Titanic_cry.copy()
Titanic_cry_nmc = Titanic_cry_nmc.dropna(how='any', subset=['I have cried during
# dividing the dataset into five subsets by the viewer's tendency to cry during
Titanic_cry1 = Titanic_cry_nmc[Titanic_cry_nmc.iloc[:,1].astype(int) == 1]
Titanic_cry2 = Titanic_cry_nmc[Titanic_cry_nmc.iloc[:,1].astype(int) == 2]
Titanic_cry3 = Titanic_cry_nmc[Titanic_cry_nmc.iloc[:,1].astype(int) == 3]
Titanic_cry4 = Titanic_cry_nmc[Titanic_cry_nmc.iloc[:,1].astype(int) == 4]
Titanic_cry5 = Titanic_cry_nmc[Titanic_cry_nmc.iloc[:,1].astype(int) == 5]

# getting rid of missing movie ratings element-wise
Titanic_cry1 = Titanic_cry1.iloc[:,0].dropna()
Titanic_cry2 = Titanic_cry2.iloc[:,0].dropna()
Titanic_cry3 = Titanic_cry3.iloc[:,0].dropna()
Titanic_cry4 = Titanic_cry4.iloc[:,0].dropna()
Titanic_cry5 = Titanic_cry5.iloc[:,0].dropna()

# Kruskal Wallis test for a non-parametric test for more than two groups
u15,p15 = stats.kruskal(Titanic_cry1,Titanic_cry2,Titanic_cry3,Titanic_cry4,Tita
print(p15 < alpha) # reject null hypothesis
len(Titanic_cry5)
```

```
True
```

Out[627… 231

In [ ]: