In [1]:
```
version = "REPLACE_PACKAGE_VERSION"
```

Experiment Design and Analysis

School of Information, University of Michigan

Week 3:

- 1. Power & Sample Sizes
- 2. Randomization - Blocking & Clustering
- 3. Differences-in-Differences

Assignment Overview

The objective of this assignment is to:

- Applying theory of experiment design and knowledge of analysis techniques to real experiment data.

The total score of this assignment will be 18 points

Resources:

- StatsModels and Scipy.stats
  - We recommend using two python libraries called StatsModels (https://www.statsmodels.org/stable/index.html) and scipy.stats (https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html) for data analysis
- Datasets used for this assignment:
  - MovieLens Data: assignment3_data.csv (assets/assignment3_data.csv)
    - Source for dataset: Chen, Y. et al. Social Comparisons and Contributions to Online Communities: A Field Experiment on MovieLens. (2010). (https://www-jstor-org.proxy.lib.umich.edu/stable/27871259)

In [2]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.stats.api as sms
import math as math
from scipy import stats
from statsmodels.stats.power import TTestIndPower
#you may or may not use all of the above libraries, and that is OK!

movie_data = pd.read_csv('assets/assignment3_data.csv') #Data for this a
```

In [3]:
```
#uncomment the below line to view readme files for this dataset (include
!cat assets/assignment3_data_readme.md

#uncomment the below line to view snippet of csv file
movie_data.tail()
```

```
### Assignment Topic: Data analysis of a field experiment on MovieLens

### Background:
We upload data files from a field experiment conducted on MovieLens, w
hich was mentioned in lecture this week.

### Data:

The MovieLens data ([assignment3_data.csv](assignment3_data.csv)) has
the following variables:

- userid: user ID

- expcondition: experiment condtion: control group or treatment group

- compare_w_median: the total number of movie a user has rated compare
d with the group median. "1": above median, "0": about median, "-1": b
elow median.

- ratings_lifetime: the total number of movie a user has rated before
experiment

- edu_year: length of education in years

- pre_rating: the number of movie a user has rated in one month before
the experimental intervention

- post_rating: the number of movie a user has rated in one month after
the experimental intervention

- active: A user is active if she rated movies, updated the database,
or invited a buddy during the two-month period of data collection. "1"
```

```
means active,  0  means not active.

- male: "1" means male, "0" means female.

- weeks: the total number of weeks a user has been using MovieLens.

- control: "1" is the control group, "0" is the treatment group.
```

Out[3]:

	userid	expcondition	userage	compare_w_median	ratings_lifetime	pre_rating	post_rating
263	151564	control	new	-1	94	90	0
264	151568	control	new	0	243	206	5
265	151579	control	new	0	121	110	0
266	151584	control	new	0	277	193	0
267	152230	conformity	new	-1	201	0	0

Introduction

In `movie_data` , you will find a dataframe containing a portion of the data from MovieLens experiment. To simply this assignment, you will only find one treatment condition where the experimenters tested the impact of social influence on moving ratings. This treatment was administrated through sending a tailored email that emphasized social influence. In contrast, subjects in the control received a plain version of email.

Part A (6 points)

First, we should check if our sample is relatively balanced across our treatment and control groups. Test the following hypotheses using a t-test:

1. The number of ratings in the month before the intervention (`pre_rating`) are balanced between the treatment and control groups. (3 points)

Round any calculations to the hundredth decimal. Do not use percentages.

In [4]:
```
def pre_ratings(provided_data):

    """
    Write the function to manually check the differences in means of pre-
    Your function should output a named dataframe with the following colu
    The dataframe should be named, 'Difference in Means between Pre-Ratin
    Tip: you can choose to use either the statsmodels stats library or tl
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    a = stats.ttest_ind(provided_data[provided_data['control'] == 1]['pre
    avg_control = round(provided_data[provided_data['control'] == 1]['pre
    avg_treatment = round(provided_data[provided_data['control'] == 0]['p
    t = round(a[0],2)
    pvalue = round(a[1],2)
    pr_df = pd.DataFrame(columns = ['avg control', 'avg treatment', 't-st
    pr_df.loc[len(pr_df.index)] = [avg_control,avg_treatment,t,pvalue]

    pr_df.name = 'Difference in Means between Pre-Rating Groups'

    return pr_df
```

Your function should return a named dataframe with each of the variables and their completed statistics. Check that it does:

In [5]:
```
pre_ratings(movie_data)
```

Out[5]:

	avg control	avg treatment	t-statistic	p-value
0	18.14	19.17	-0.09	0.93

In [6]:
```
assert pre_ratings(movie_data).name == "Difference in Means between Pre-R
df_columns = ['avg control', 'avg treatment', 't-statistic', 'p-value']
for index, title in enumerate(pre_ratings(movie_data).columns):
    assert title == df_columns[index]
```

In [7]:
```
"""Checking avg_control and avg_treatment values"""
# Hidden tests
```

Out[7]: 'Checking avg_control and avg_treatment values'

In [8]:
```
"""checking your t-statistic and p-value are correct"""
# Hidden tests
```

Out[8]: 'checking your t-statistic and p-value are correct'

2. Test that the gender composition (variable 'male') is similar between the treatment and control groups. (3 points)

Round any calculations to the hundredth decimal. Do not use percentages.

In [9]:
```
def male_gender_comp(provided_data):

    """
    Write the function to manually check the differences in means of part
    Your function should output a named dataframe with the following colu
    The dataframe should be named, 'Difference in Means of Males'.
    Tip: you can choose to use either the statsmodels stats library or tl
    """
    gender_control = movie_data[movie_data['control'] == 1]['male']
    # YOUR CODE HERE
    #raise NotImplementedError()


    a = stats.ttest_ind(provided_data[provided_data['control'] == 1]['mal
    avg_control = round(provided_data[provided_data['control'] == 1]['mal
    avg_treatment = round(provided_data[provided_data['control'] == 0]['m
    t = round(a[0],2)
    pvalue = round(a[1],2)

    male_df = pd.DataFrame(columns = ['avg control', 'avg treatment', 't-
    male_df.loc[len(male_df.index)] = [avg_control,avg_treatment,t,pvalue

    male_df.name = 'Difference in Means of Males'

    return male_df
```

Your function should return a named dataframe with each of the variables and their completed statistics. Check that it does:

In [10]:
```
male_gender_comp(movie_data)
```

Out[10]:

	avg control	avg treatment	t-statistic	p-value
0	0.55	0.58	-0.49	0.62

In [11]:
```
"""Checking the t-statistic and p-value in your dataframe are correct"""
assert next(iter(male_gender_comp(movie_data)['t-statistic'])) == -0.49
assert next(iter(male_gender_comp(movie_data)['p-value'])) == 0.62
```

In [12]:
```
"""Part A #2: Checking your dataframe is named, and your columns are in c
# Hidden tests
```

Out[12]:
```
'Part A #2: Checking your dataframe is named, and your columns are in
order'
```

In [13]:
```
"""Part A #2: Checking your avg control and avg treatment values are cor
# Hidden tests
```

Out[13]:
```
'Part A #2: Checking your avg control and avg treatment values are cor
rect'
```

Part B (6 points)

From the MovieLens experiment, we know that we want to estimate the impact of social influence on moving ratings on the MovieLens platform. Let's estimate this by using difference-in-differences to examine the effects of post_rating for the treatment and control group.

1. Create a new variable, delta, in the dataframe and output the dataframe. Delta should show the difference in pre_rating and post_rating (calculate using post_rating – pre_rating). (2 points)

In [14]:
```
def delta_ratings(provided_data):

    """
    Write the function to output a new dataframe with the following colu
    The content of the columns should come from movie_data. Delta should
    The dataframe should be named, 'Delta in Ratings'.
    """
    # YOUR CODE HERE
    #raise NotImplementedError()

    delta_ratings_df = provided_data[['userid','compare_w_median','pre_ra
    delta_ratings_df['delta'] = delta_ratings_df['post_rating'] - delta_r

    delta_ratings_df = delta_ratings_df[['userid','compare_w_median','pre
    delta_ratings_df.name = 'Delta in Ratings'


    return delta_ratings_df
```

Your function should return a named dataframe with each of the variables and their values. Check that it does:

In [15]:
```
delta_ratings(movie_data).head()
```

Out[15]:

	userid	compare_w_median	pre_rating	post_rating	delta	control
0	42126	1	0	0	0	1
1	47947	0	0	38	38	1
2	49034	-1	0	41	41	1
3	51898	0	0	8	8	0
4	52797	0	0	33	33	0

In [16]:
```
"""checking you have a named dataframe"""
assert delta_ratings(movie_data).name == "Delta in Ratings"
```

In [17]:
```
"""checking your column names and orders"""
# Hidden tests
```

Out[17]:
```
'checking your column names and orders'
```

2. Use an ordinary least squares regression model to explore the average treatment-effect on delta. Using a t-test, what is the significance using the t-statistic and p-value of this effect? (4 points)

Round any calculations to the hundredth decimal. Do not use percentages.

In [18]:
```
import statsmodels.formula.api as smf

def ate_delta_avg(provided_data):

    """ The easiest way to evaluate the average treatment effect is to ru
    the dependent variable, and control as the independent variable. Use
    OLS linear regression, and return a named dataframe with the t-statis
    data.
    The dataframe should have the following columns: 't-statistic', 'p-va
    Effect on Delta'
    """
    # complete the function by assigning your X and Y, and fitting your m
    # YOUR CODE HERE
    #raise NotImplementedError()

    provided_data = delta_ratings(provided_data)

    model = smf.ols(formula = "provided_data['delta'] ~ provided_data['co

    tstats = model.tvalues

    pvals = model.pvalues

    pval = np.round(pvals[1],2)

    tstat = np.round(tstats[1],2)

    did_df = pd.DataFrame(data = [[tstat, pval]], columns=['t-statistic',
    did_df.name = 'Average Treatment Effect on Delta'

    return did_df
```

Your function should return a named dataframe with the correct values. Check that it does:

In [19]:
```
ate_delta_avg(movie_data) #again, if you get a deprecation warning, that
```

Out[19]:

	t-statistic	p-value
0	-0.49	0.62

In [20]:
```
"""Part B #2: Checking your dataframe is named, and your columns are in c
# Hidden tests
```

Out[20]:
```
'Part B #2: Checking your dataframe is named, and your columns are in
order'
```

In [21]:
```
"""checking your t-statistic and p-value are correct"""
# Hidden tests
```

Out[21]:
```
'checking your t-statistic and p-value are correct'
```

Part C (8 points)

What if we break this comparison down by group, specifically the total number of ratings users complete compared with the median ratings (compare_w_median)?

1. Output the t-statistics and p-values for the average treatment-effect on delta across median ratings (where `compare_w_median == -1`, where `compare_w_median == 0`, and where `compare_w_median == 1`). (8 points)

In [22]:
```
def ate_delta_median_values(provided_data):
    """ The easiest way to evaluate the average treatment effect is to ru
    distinct 'compare_w_median' values, with delta as the dependent varia
    variable. Use the statsmodels library to run OLS linear regressions,

    The dataframe should be indexed, with the index values as follows: 'b
    The dataframe should have the following columns: 't-statistic', 'p-va
    The dataframe should be named 'Average Treatment Effect across Media
    """

    # YOUR CODE HERE
    #raise NotImplementedError()
    df = pd.DataFrame(columns=['t-statistic', 'p-value'], index = ['belo
    df_org = provided_data

    for i,j in zip([-1,0,1],['below median', 'at median', 'abv median']):
        provided_data = df_org
        provided_data = delta_ratings(provided_data[provided_data['compa

        model = smf.ols(formula = "provided_data['delta'] ~ provided_data

        tstats = model.tvalues

        pvals = model.pvalues

        pval = np.round(pvals[1],2)

        tstat = np.round(tstats[1],2)
        df.loc[j]['t-statistic'] = tstat
        df.loc[j]['p-value'] = pval

    df.name='Average Treatment Effect across Median Scores'
    return df
```

Your function should return a named and indexed dataframe with the correct values. Check that it does:

In [23]:
```
ate_delta_median_values(delta_ratings(movie_data))
```

Out[23]:

	t-statistic	p-value
below median	-2.45	0.02
at median	-2.19	0.03
abv median	0.71	0.48

In [24]:
```
"""checking your dataframe is named, your columns are in order, and you h
assert ate_delta_median_values(delta_ratings(movie_data)).name == 'Averag
check_col = iter(ate_delta_median_values(delta_ratings(movie_data)).colum
check_ind = iter(ate_delta_median_values(delta_ratings(movie_data)).index
assert next(check_ind) == 'below median'
assert next(check_ind) == 'at median'
assert next(check_ind) == 'abv median'
assert next(check_col) == 't-statistic'
assert next(check_col) == 'p-value'
```

In [25]:
```
"""checking your below-median t-statistic and p-value are correct"""
# Hidden tests
```

Out[25]: 'checking your below-median t-statistic and p-value are correct'

In [26]:
```
"""checking your at-median t-statistic and p-value are correct"""
# Hidden tests
```

Out[26]: 'checking your at-median t-statistic and p-value are correct'

In [27]:
```
"""checking your abv-median t-statistic and p-value are correct"""
# Hidden tests
```

Out[27]: 'checking your abv-median t-statistic and p-value are correct'

Part D (5 points)

Based on the `post_rating` observations, perform a sample-size calculation to determine the minimum number of subjects that are needed for detecting difference in means between the treament and the control groups. Assume $\alpha = 0.05$ and $\beta = 0.1$, and that the variances are the same for the treatment and control groups.

The `solve_power` method of the `TTestIndPower` class provided by `statsmodels` can be used to solve this problem. You may want to carefully read its [documentation] (https://www.statsmodels.org/stable/generated/statsmodels.stats.power.TTestIndPower.solve_po (https://www.statsmodels.org/stable/generated/statsmodels.stats.power.TTestIndPower.solve_po to understand how to use it. Use `TTestIndPower().solve_power` to call the function. Make sure to round up the number of observations by using math.ceil(....).

You'd also need to understand the difference between *population* and *sample* standard deviation, and how to use `pd` functions to calculate either one.

In [28]:
```
from numpy import std, mean, sqrt

def power_calc(provided_data):
    """
    Your function should return a named pd.Series, "Power Analysis", that
     - ctrl_mean: the mean for the control group
     - trtm_mean: the mean for the treatment group
     - pop_std: the population standard deviation for both the control ar
     - num_obs: the minimum number of subjects required
    """
    # YOUR CODE HERE
    #TTestIndPower.solve_power(effect_size=_effect_size, nobs1=None, alpl
    control = provided_data[provided_data['control'] == 1]
    treat = provided_data[provided_data['control'] == 0]
    ctrl_mean = np.mean(control['post_rating'])
    trtm_mean = np.mean(treat['post_rating'])
    pop_std = np.std(provided_data['post_rating'])
    effect= (ctrl_mean-trtm_mean)/std(movie_data['post_rating'])

    alpha = 0.05

    power = 0.9

    ratio = len(control['post_rating'])/len(treat['post_rating'])
    analysis = TTestIndPower()
    result = analysis.solve_power(effect_size = effect, alpha=alpha, powe
    num_obs = math.ceil(result)

    index_labels = ['ctrl_mean', 'trtm_mean', 'pop_std', 'num_obs']
    ser = pd.Series(data = [ctrl_mean, trtm_mean, pop_std, num_obs], inde
    ser.name = "Power Analysis"

    return ser

    #raise NotImplementedError()
```

In [29]:
```
# Check your result
power_calc(movie_data)
```

Out[29]:
```
ctrl_mean     12.179104
trtm_mean     19.074627
pop_std       34.197445
num_obs      518.000000
Name: Power Analysis, dtype: float64
```

In [30]:
```
# Visible tests

stu_ans = power_calc(movie_data)

assert isinstance(stu_ans, pd.Series), "Part D: Your function should retu
assert stu_ans.name == "Power Analysis", "Part D: Your Series should be r
assert list(stu_ans.index) == ['ctrl_mean', 'trtm_mean', 'pop_std', 'num_

del stu_ans
```

In [31]:
```
# Hidden tests
```

In [ ]:
```

```