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

Experiment Design and Analysis

School of Information, University of Michigan

Week 2:

Assignment Overview

The objective of this assignment is to:

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

The total score of this assignment will be 19 points

Resources:

- StatsModels, Scipy.stats, Numpy
  - We recommend using two python libraries called StatsModels and Scipy.stats for data analysis. For this dataset, we'll be using Numpy as well.
- Optional Reading: Holt C.A, & Laury S.K. Risk Aversion and Incentive Effects. (2002).

Datasets used in this assignment

- Trust data download csv file
- Fixed-Price Auction data download csv file
  - Source for dataset: Chen, Y., et al. Sealed bid auctions with ambiguity: Theory and experiments. (2007).

In [ ]:

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

trust_data = pd.read_csv('assets/assignment2_data1.csv') #Trust Game data f
fpa_data = pd.read_csv('assets/assignment2_data2.csv') #First-price auction
```

In [ ]:

```
#uncomment the below line to view readme files for this dataset (includes e
#!cat assets/assignment2_data1_readme.md
#!cat assets/assignment2_data2_readme.md

#uncomment the below line to view snippet of csv file
#trust_data.head()
#fpa_data.head()
```

Part A (3 points)

1. For the Trust Game, subjects are grouped in pairs, paired with one assigned the role of an investor and another a recipient. Let's examine the correlation between the amounts the investors invest and the amounts the recipients return. Complete the function below to return the correlation coefficient.

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

In [ ]:

```
def inv_rec_corrcoef(provided_data):
    """ Later in this problem set, you will be modeling OLS regressions on 
    the correlation coefficient using numpy. If needed, refer to the numpy 
    """
    ### BEGIN SOLUTION
    invest = trust_data[trust_data['player role'] == 'first']['decision']
    returner = trust_data[trust_data['player role'] == 'second']['decision'
    answer = round(np.corrcoef(invest, returner)[0][1],2)
    ### END SOLUTION
    return answer
```

Your function should return a string with the correct coefficient value. Check that it does:

In [ ]:

```
inv_rec_corrcoef(trust_data)
```

In [ ]:

```
assert type(inv_rec_corrcoef(trust_data)) == np.float64
```

In [ ]:
```
"""Part A #1: Checking value of correlation coefficient"""
# Hidden tests
### BEGIN HIDDEN TESTS
assert inv_rec_corrcoef(trust_data) == 0.82, "Part A #1 correlation coeffic
### END HIDDEN TESTS
```

Part B (4 points)

For the first-price auctions experiment, there are ten experimental sessions, with eight subjects per session. In this context, subjects are tasked with completing auction and lottery (Holt-Laury 2002) tasks in two orders. In five of the ten sessions, subjects first complete a lottery task, followed by 30 rounds of auctions. In the other five sessions, subjects first complete 30 rounds of auctions, followed by a lottery task. At the end of each session, subjects complete a demographics survey. The data sets extract the first period auction data for each treatment.

In this case, say that the control for the first-price auction experiment is the order in which subjects complete the lottery task followed by the auction task (k1_8_lot_exp) and the outcome variable we want to measure is the bid-value ratio (b/v).

1. Using differences-in-means, what is the average treatment effect for the first-price auction experiment? (4 points)

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

In [ ]:
```
def ate_fpa_payoff(provided_data):
    """
    Write the function to manually check the differences in means of bid-va
    Tip: the easiest way to do this is to create a new dataframe column cal
    Your function should output a dataframe with the following columns: 'lo
    """
    ### BEGIN SOLUTION
    ate_fpa_payoff_df = pd.DataFrame(columns=['lot_auc_mean','auc_lot_mean'
    provided_data['bidval_ratio'] = provided_data['b'] / provided_data['v']
    ate_fpa_payoff_df['lot_auc_mean'] = [round(provided_data[provided_data[
    ate_fpa_payoff_df['auc_lot_mean'] = [round(provided_data[provided_data[
    ate_fpa_payoff_df['diff in means'] = ate_fpa_payoff_df['lot_auc_mean']-
    ###END SOLUTION
    return ate_fpa_payoff_df
```

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

In [ ]:
```
ate_fpa_payoff(fpa_data)
```

In [ ]:
```
assert isinstance (ate_fpa_payoff(fpa_data), pd.core.frame.DataFrame), "che
```

In [ ]:
```
assert ate_fpa_payoff(fpa_data).columns[0] == 'lot_auc_mean', "checking df
assert ate_fpa_payoff(fpa_data).columns[1] == 'auc_lot_mean', "checking df
assert ate_fpa_payoff(fpa_data).columns[2] == 'diff in means', "checking df
```

In [ ]:
```
"Part B: lot_auc_mean value"
# Hidden tests
### BEGIN HIDDEN TESTS
assert next(iter(ate_fpa_payoff(fpa_data)["lot_auc_mean"])) == round(fpa_da
### END HIDDEN TESTS
```

In [ ]:
```
"Part B: auc_lot_mean value"
# Hidden tests
### BEGIN HIDDEN TESTS
assert next(iter(ate_fpa_payoff(fpa_data)['auc_lot_mean'])) == round(fpa_da
### END HIDDEN TESTS
```

Part C (10 points)

Continuing with the `fpa_data` dataset from last week, we would expect subjects to bid a certain fraction of their value in a first-price sealed bid auction depending on their risk attitudes (e.g., risk neutral, risk averse). Let's explore what effect gender has on bid-value ratios when controlled with risk. This time, let's calculate this average treatment effect using an ordinary least-squares regression.

1. Using the `fpa_data` dataframe and an ordinary least-squares regression model, complete the `ols_riskfemale_on_bidvalue` function to evaluate how subjects' risk attitudes and gender (in the form of the *female* variable) affect their bid/value ratio. For now, we'll just return a summary view of our data. (2 points)

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

In [ ]:

```
def ols_riskgender_on_bidvalue(provided_data):

    """
    The easiest way to evaluate how subjects' risk attitudes and gender aff
    regression on your fpa_data dataframe. Use the statsmodels library to r
    view of your results.

    """
    X = provided_data[['female', 'ra']]
    ### BEGIN SOLUTION
    Y = provided_data['bv_ratio'] = provided_data['b'] / provided_data['v']
    X = sm.add_constant(X)
    model = sm.OLS(Y,X).fit()
    model_summary = model.summary()
    ### END SOLUTION
    return model_summary
```

Your function should return a summary view of your results. Check that it does:

In [ ]:

```
print(ols_riskgender_on_bidvalue(fpa_data)) #we've wrapped this in a print
```

In [ ]:

```
assert isinstance (ols_riskgender_on_bidvalue(fpa_data), sm.iolib.summary.S
```

1. Now, modify the ols_riskgender_on_bidvalue function to access the model's coefficients (parameters) and associated p-values, instead of printing out the entire summary view. For now, we won't worry about rounding. (1 point)

In [ ]:

```
def ols_riskgender_on_bidvalue(provided_data):

    """
    The easiest way to evaluate how subjects' risk attitudes and gender aff
    regression on your data dataframe. Use the statsmodels library to run a
    the coefficients and the p-values for your model.

    """
    X = provided_data[['female', 'ra']]
    # complete the function by assigning your Y, and fitting your model.
    ### BEGIN SOLUTION
    Y = provided_data['bv_ratio'] = provided_data['b'] / provided_data['v']
    X = sm.add_constant(X)
    model = sm.OLS(Y,X).fit()
    model_params = model.params
    pvals = model.pvalues
    ### END SOLUTION
    return model_params, pvals #we're returning a tuple of a series here --
```

Your function should return a raw tuple of your results in pandas Series form. Check that it does:

In [ ]:
```
ols_riskgender_on_bidvalue(fpa_data)
```

In [ ]:
```
"checking your return value is a tuple of type pandas series"
assert isinstance (ols_riskgender_on_bidvalue(fpa_data)[0], pd.core.series.
assert isinstance (ols_riskgender_on_bidvalue(fpa_data)[1], pd.core.series.
```

In [ ]:
```
"checking the value of 'const' for both values"
# Hidden tests
### BEGIN HIDDEN TESTS
assert round(next(iter(ols_riskgender_on_bidvalue(fpa_data)[0])),2) == 1.06
assert round(next(iter(ols_riskgender_on_bidvalue(fpa_data)[1])),2) == 0.07
### END HIDDEN TESTS
```

1. Now, let's make our results more readable. Let's modify our function once again to this time create a dataframe that has the coefficients and p-values for the control variables and constant, **rounding to the nearest hundredth decimal**. (2 points)

In [ ]:
```
def ols_riskgender_on_bidvalue_df(provided_data):

    """
    This function should use the results of the ols_riskgender_on_bidvalue
    that has the coefficients and p-values for the control variables and co
    The dataframe should have the following columns: 'variable', 'coefficie

    """
    # define your parameters for your model and the p-values, then fill in

    ### BEGIN SOLUTION
    model_params = ols_riskgender_on_bidvalue(provided_data)[0]
    pvals = ols_riskgender_on_bidvalue(provided_data)[1]
    ### END SOLUTION
    ols_model_df = pd.DataFrame(columns=['variable','coefficient','p-value'
    variables = ['const','ra','female']
    ols_model_df['variable'] = variables
    for variable in ols_model_df['variable']:
        ### BEGIN SOLUTION
        ols_model_df.loc[ols_model_df['variable']== variable,'coefficient']
        ols_model_df.loc[ols_model_df['variable']==variable,'p-value'] = ro
        ### END SOLUTION
    return ols_model_df
```

Your function should return a dataframe of your results. Check that it does:

In [ ]:
```
ols_riskgender_on_bidvalue_df(fpa_data)
```

In [ ]:
```
"""Part C: Check the dataframe outputs the correct p-values from OLS model"
# Hidden tests
### BEGIN HIDDEN TESTS
assert ols_riskgender_on_bidvalue_df(fpa_data).iloc[2][2] == 0.41, "Part C
assert ols_riskgender_on_bidvalue_df(fpa_data).iloc[1][2] == 0.89, "Part C
### END HIDDEN TESTS
```

In [ ]:
```
"""Part C: Check the dataframe outputs the correct coefficients from OLS mo
# Hidden tests
### BEGIN HIDDEN TESTS
assert ols_riskgender_on_bidvalue_df(fpa_data).iloc[2][1] == -0.21, "Part C
assert ols_riskgender_on_bidvalue_df(fpa_data).iloc[1][1] == -0.01, "Part C
### END HIDDEN TESTS
```

1. If you remove the risk attitudes variable from the model, does it have a significant effect on how gender contributes to bid/value ratios? Complete the `ols_female_on_bidvalue` function to assess this. Part of the function has already been completed for you. (3 points)

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

In [ ]:

```
def ols_gender_on_bidvalue_df(provided_data):

    """
    Complete the function that takes the provided data and creates a OLS mo
    gender (using the female variable) on subjects' bid/value ratios. It sh
    a dataframe that has the coefficients for the control variables and int

    """
    # assign your X and Y variables, and define your parameters and pvalues
    ### BEGIN SOLUTION
    X = provided_data['female']
    Y = provided_data['bv_ratio'] = provided_data['b'] / provided_data['v']
    X = sm.add_constant(X)
    model = sm.OLS(Y,X).fit()
    model_params = model.params
    pvals = model.pvalues
    ### END SOLUTION
    ols_gender_model_df = pd.DataFrame(columns=['variable','coefficient','p
    variables = ['const','female']
    ols_gender_model_df['variable'] = variables

    for variable in ols_gender_model_df['variable']:
        ### BEGIN SOLUTION
        ols_gender_model_df.loc[ols_gender_model_df['variable']==variable,'
        ols_gender_model_df.loc[ols_gender_model_df['variable']==variable,'
        ### END SOLUTION
    return ols_gender_model_df
```

Your function should return a dataframe with each of the variables and their completed coefficient and p-value for the OLS model.

Check that it does:

In [ ]:

```
ols_gender_on_bidvalue_df(fpa_data)
```

In [ ]:

```
assert ols_gender_on_bidvalue_df(fpa_data).iloc[0][2] == 0, "checking the c
```

In [ ]:

```
"""Check that the dataframe outputs the correct values from the OLS model""
# Hidden tests
### BEGIN HIDDEN TESTS
assert ols_gender_on_bidvalue_df(fpa_data).iloc[0][1] == 0.98, "Part C cons
assert ols_gender_on_bidvalue_df(fpa_data).iloc[1][1] == -0.21, "Part C fem
assert ols_gender_on_bidvalue_df(fpa_data).iloc[1][2] == 0.4, "Part C femal
### END HIDDEN TESTS
```