Stats Package Instruction Manual

Table of Contents:

1. Setup

a. Prerequisites

i. In order to get started with the stats package you need to have python and jupyter notebooks set up on your computer. Follow the instructions under “initial setup” here if you don’t already have those installed.

b. Installing the Stats Package

i. Close any open jupyter notebooks, open terminals, and anaconda
ii. Open your terminal on your desktop (cmd+space, type “terminal”)
iii. Copy and paste the following in the terminal and press enter:
    1. pip install "git+https://github.com/ejones-rescue/Stats_Package.git"
iv. If it was installed correctly it will say “Successfully installed Stats-Package” at the end of the load
    1. If it asks you if you want to install any dependencies type Y and enter to continue
    2. If you get any other errors during installation reach out to Elisabeth

c. Importing the Stats Packages to python

i. Every time you create a new python file that you want to use the Stats Packages in you need to import them using these two lines of code (copy and paste into your code):
    1. from Stats_Package.stats_descriptives_module import freqs, desc, desc_subcats, xtab_cols, xtab_rows, xtab_all
    2. from Stats_Package.stats_analysis_module import chi_sq, chisq_posthoc_bon, ttest_1sample, ttest_2sample, ttest_paired, KW_shape_assumption, corr_pearson
    3. Import pandas as pd
    4. Import numpy as np

d. Before you get started

i. Before you get started on using the stats package there are a few essential python things to know, information is listed below but there is also example python code and base templates that can help you in this folder.
    1. **Reading in your data file**
        a. In order to read in you data file to python you will need to copy, edit, and run the following code:
            i. df = pd.read_excel(‘name of your file.xlsx’)
                1. If your file is in excel format and located in the folder you're working out of
            ii. df = pd.read_csv(‘name of your file.csv’)

1. If your file is in csv format and located in the folder you're working out of

2. **Exporting your stats tables to excel**
    a. In order to export your tables you will need to copy, edit, and run the following code:
        i. your_table.to_excel('your_table_name.xlsx', index = False)
            1. This will save in the current folder you are working out of
            2. To save into a different folder you can use
                a. your_table.to_excel('../your_table_name.xlsx', index = False)
                    i. This will save in the file above the one you are working out of
                b. your_table.to_excel('../another folder name/your_table_name.xlsx', index = False)
                    i. This will navigate to the file above your current folder and into a different subfolder to save your file

3. **Jupyter Notebook tips**
    a. Jupyter notebook automatically saves your work, but to be safe you can click on File > Save and Checkpoint to save any progress.
    b. Sometimes you'll want to clear out the outputs within your jupyter notebook and start fresh. To do this click on Kernal > Restart & Clear Output
    c. To make comments on your code use a hashtag #
    d. To create a markdown cell create a new cell as hit esc+m. Here's a markdown cheatsheet for your reference.

4. **Cleaning your data**
    a. Please ensure your data is fully cleaned before using the stats package. Feel free to review the Data Cleaning SOP if you're unsure of where to start.
    b. One important check to do before using the stats package is to ensure that your missing data is truly missing, not a space or 999, but just blank. If your missing cases are not truly missing they will be included in the various calculations within the stats package when they should be excluded.

2. Stats Descriptives Functions (Use for basic descriptives of your data)

a. Frequencies | freqs()

i. Use the freqs() function to export counts and percentages for any given variables. This function will work best with categorical data.

ii. Parameters:
    1. freqs(df, [list of variables], True/False)
        a. df = name of your data frame
        b. [list of variables] = list of the names of variables you want to run frequencies on
            i. Each of your variable names must be in quotes and separated by commas
            ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets.
        c. True/False = enter False if you want NA/missing cases to be included, enter True if you want NA/missing cases to be excluded
iii. Examples:
    1. Multiple variables:
        a. frequency_demos = freqs(df,[‘Age_Categories’, ‘Race’, ‘Gender’, ‘Region’], True)
            i. This would create a table called frequency_demos that has frequencies for Age_Cateogories, Race, Gender, and Region. It would not include missing cases from any of the variables.
    2. Single variable:
        a. frequency_gender = freqs(df, [‘Gender’], True)
            i. This would create a table called frequency_gender that has frequencies on Gender only. Any missing cases from Gender would be included.

b. Descriptives | desc()

i. Use the desc() function to export the count, mean, standard deviation, min, interquartile range (25%, 50%, and 75%, where 50% is the median), and max for any given variables. This function will work best with numerical/continuous variables.
ii. Parameters:
    1. desc(df, [list of variables])
        a. df = name of your data frame
        b. [list of variables] = list of the names of variables you want to run descriptives on
            i. Each of your variable names must be in quotes and separated by commas
            ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
            iii. Note that NA/missing cases will automatically be excluded from the desc() function

iii. Examples:
    1. Multiple variables:
        a. desc_p30dayuse = desc(df, [‘P30D_Cigs’, ‘P30D_Vapes’, ‘P30D_Cigars’])
            i. This would create a table called desc_p30dayuse that has descriptives for past 30 day use of cigarettes, vapes, and cigars.
    2. Single variable:
        a. desc_vapes = desc(df, [‘P30D_Vapes’])
            i. This would create a table called desc_vapes that has descriptives of the past 30 day use vape variable.

c. Sub-Category Descriptives | desc_subcats()

i. Use the desc_subcats() function to run descriptives (count, mean, standard deviation, min, interquartile range, and max) for any given list of variables separated out by another subcategory variable. This function will work best with numerical/continuous variables.

ii. Parameters:
    1. desc_subcats(df, [list of variables], subcategory_variable, [list of subcategories])
        a. df = name of your data frame
        b. [list of variables] = list of the names of variables you want to run descriptives on
            i. Each of your variable names must be in quotes and separated by commas
            ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
            iii. Note that NA/missing cases will automatically be excluded from the desc_subcats() function
        c. subcategory_variable = the name of the variable you want to separate out descriptives by
            i. This can only be one variable
            ii. It must be in quotes
        d. [list of subcategories] = the labels of each of the subcategories within your subcategory_variable
            i. If the subcategory labels are strings (i.e., words or characters) they must be in quotes
            ii. If the subcategory labels are numbers do not use quotes
            iii. Each label must be separated by a comme regardless of if they are strings or numbers

iii. Examples:

1. If I want to run descriptive on multiple past 30 day use variables separated by gender (where 1 = Female, and 2 = Male) I would do the following:
    a. desc_p30d_gender = desc_subcats(df, [‘P30D_Cigs’, ‘P30D_Vapes’, ‘P30D_Cigars’], ‘Gender’, [1,2])
2. If I want to run descriptives on age separated by peer crowd (i.e., what is the average age per each peer crowd?) I would do the following:
    a. desc_age_pc = desc_subcats(df, [‘Age’], ‘Exclusive_PC’, [‘Hip Hop’, ‘Alternative, ‘Mainstream’, ‘Popular’, ‘Country’])

d. Cross tabs w/ column percentages | xtab_cols()

i. Use the xtab_cols() function to create a cross tabulation of any given variables by another single variable. This function will produce percentages out of the *total of the columns*. This function works best with categorical variables. To determine if there is significance in your crosstab you can use the chi_sq() function.

ii. Parameters:
    1. xtab_cols(df, [list of variables], column_variable)
        a. df = name of your data frame
        b. [list of variables] = list of the names of variables you want to run, these will populate the rows of the crosstab tables
            i. Each of your variable names must be in quotes and separated by commas
            ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
            iii. Note that NA/missing cases will automatically be excluded from the function unless your data is not cleaned properly (see "cleaning your data")
        c. column_variable = name of the variable you want to split the data by, this variable will populate the columns of the crosstab tables.
            i. The variable name must be in quotes

iii. Examples:
    1. If I want to run a crosstab of multiple demographic variables by awareness using column percentages I would do the following:
        a. xtab_demos_aware = xtab_cols(df, [‘Race’, ‘Region’, ‘Gender’], ‘Campaign_Awareness’)
            i. This would produce a table called xtab_demos_aware that contains columns split by Campaign_Awareness and rows split by Race, Region, and Gender.

2. If I want to run a crosstab of peer crowds by Age_Cateogries using column percentages I would do the following:
    a. xtab_pc_age = xtab_cols(df, [‘Exclusive_PC’], ‘Age_Categories’)
        i. This would produce a table called xtab_pc_age that contains columns split by Age_Categories, and rows split by peer crowds.

e. Cross tabs w/ row percentages | xtab_rows()

i. Use the xtab_rows() function to create a cross tabulation of any given variable(s) by another single variable. This function will produce percentages out of the *total of the rows.* This function works best with categorical variables. To determine if there is significance in your crosstab you can use the chi_sq() function.

ii. Parameters:
    1. xtab_rows(df, [list of variables], column_variable)
        a. df = name of your data frame
        b. [list of variables] = list of the names of variables you want to run, these will populate the rows of the crosstab tables
            i. Each of your variable names must be in quotes and separated by commas
            ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
            iii. Note that NA/missing cases will automatically be excluded from the function unless your data is not cleaned properly (see "cleaning your data")
        c. column_variable = name of the variable you want to split the data by, this variable will populate the columns of the crosstab tables.
            i. The variable name must be in quotes

iii. Examples:
    1. If I want to run a crosstab of multiple demographic variables by awareness using row percentages I would do the following:
        a. xtab_demos_aware = xtab_rows(df, [‘Race’, ‘Region’, ‘Gender’], ‘Campaign_Awareness’)
            i. This would produce a table called xtab_demos_aware that contains columns split by Campaign_Awareness and rows split by Race, Region, and Gender.
    2. If I want to run a crosstab of peer crowds by Age_Cateogries using row percentages I would do the following:
        a. xtab_pc_age = xtab_rows(df, [‘Exclusive_PC’], ‘Age_Categories’)

i. This would produce a table called xtab_pc_age that contains columns split by Age_Categories, and rows split by peer crowds.

f. Cross tabs w/ total percentages | xtab_all()

i. Use the xtab_all() function to create a cross tabulation of any given variable(s) by another single variable. This function will produce percentages out of the *total of the entire dataset*. This function works best with categorical variables. To determine if there is significance in your crosstab you can use the chi_sq() function.

ii. Parameters:

1. xtab_all(df, [list of variables], column_variable)
    a. df = name of your data frame
    b. [list of variables] = list of the names of variables you want to run, these will populate the rows of the crosstab tables
        i. Each of your variable names must be in quotes and separated by commas
        ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
        iii. Note that NA/missing cases will automatically be excluded from the function unless your data is not cleaned properly (see "cleaning your data")
    c. column_variable = name of the variable you want to split the data by, this variable will populate the columns of the crosstab tables.
        i. The variable name must be in quotes

iii. Examples:

1. If I want to run a crosstab of multiple demographic variables by awareness using total percentages I would do the following:
    a. xtab_demos_aware = xtab_all(df, [‘Race’, ‘Region’, ‘Gender’], ‘Campaign_Awareness’)
        i. This would produce a table called xtab_demos_aware that contains columns split by Campaign_Awareness and rows split by Race, Region, and Gender.
2. If I want to run a crosstab of peer crowds by Age_Cateogries using total percentages I would do the following:
    a. xtab_pc_age = xtab_all(df, [‘Exclusive_PC’], ‘Age_Categories’)
        i. This would produce a table called xtab_pc_age that contains columns split by Age_Categories, and rows split by peer crowds.

3. Stats Analysis Functions (Use for basic analysis of your data)

a. Chi Square | chi_sq()

i. Use the chi_sq() function to compare observed results with expected results. The purpose of this test is to determine if a difference between observed data and expected data is due to chance, or if it is due to a relationship between the variables you are studying. This function should only be used when comparing categorical variables.

ii. Parameters:

1. chi_sq(df, [list of dependent variables], idependent_variable, significance_level)
    a. df = name of your data frame
    b. [list of dependent variables] = list of the names of variables of the dependent you want to run
        i. Each of your variable names must be in quotes and separated by commas
        ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
        iii. Note that NA/missing cases will automatically be excluded from the function unless your data is not cleaned properly (see "cleaning your data")
    c. Independent_variable = variable that you want split each of your dependent variables on
        i. The variable name must be in quotes
    d. significance_level = the numeric level of significance you want to evaluate p-values by.
        i. Do not use quotes
        ii. If you are unsure, it's a rule of thumb to use 0.05

iii. Examples:

1. If I want to know if multiple categorical demographic variables are significantly different based on awareness I would do the following:
    a. chisq_demos_awareness = chi_sq(df, ['Age_Categories', 'Race', 'Region', 'Gender'], 'Campaign_Awareness', 0.05)
        i. This would produce a table called chisq_demos_awareness containing a list of your dependent variables, test statistics, and a Significance column that = True if your p-value was significant at the 0.05 level)

b. Chi Square Bonferroni Post-Hoc | chisq_posthoc_bon()

i. Use the chisq_posthoc_bon() function to determine which levels were significant based on the chi_sq() test. This functions should be used as a follow up to any significant chi square analyses. This function should only be used when comparing categorical variables.

ii. Parameters:

1. chisq_posthoc_bon(df, [list of dependent variables], idependent_variable, significance_level)
    a. df = name of your data frame
    b. [list of dependent variables] = list of the names of dependent variables you want to run
        i. Each of your variable names must be in quotes and separated by commas
        ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
        iii. Note that NA/missing cases will automatically be excluded from the function unless your data is not cleaned properly (see "cleaning your data")
    c. Independent_variable = variable that you want split each of your dependent variables on
        i. The variable name must be in quotes
    d. significance_level = the numeric level of significance you want to evaluate p-values by.
        i. Do not use quotes
        ii. If you are unsure, it's a rule of thumb to use 0.05

iii. Examples:

1. After running a chi square analysis on multiple demographics variables vs awareness, results showed that Race and Region were significant - to determine which levels are driving the significance I would do the following:
    a. bon_demos_awareness = chisq_posthoc_bon(df, ['Race', 'Region'], 'Campaign_Awareness', 0.05)
        i. This would produce a table called bon_demos_awareness that prints all independent variable pairs, their corresponding dependent variable, and identifies which levels are significant at the 0.05 level.

c. Two Sample T-test | ttest_2sample()

i. Use the ttest_2sample() function to determine if there is a significant difference between two means. This function should be used with continuous dependent variables and binary (categorical or numeric) independent variables.

ii. Parameters:
    1. ttest_2sample(df, [list of dependent variables], independent_variable, [list of binary subcategories], significance_level)
        a. df = name of your data frame
        b. [list of dependent variables] = list of the names of dependent variables you want to run
            i. Each of your variable names must be in quotes and separated by commas
            ii. The list must be surrounded by square brackets, if you only want to run one variable, it must still be within square brackets
            iii. Note that NA/missing cases will automatically be excluded from the function unless your data is not cleaned properly (see "cleaning your data")
        c. Independent_variable = variable that you want split each of your dependent variables on
            i. The variable name must be in quotes
        d. [list of binary subcategories] = list of subcategories from your independent variable that you want to split on. You can only enter two categories into this list.
            i. If the subcategory labels are strings (i.e., words or characters) they must be in quotes
            ii. If the subcategory labels are numbers do not use quotes
            iii. Each label must be separated by a comme regardless of if they are strings or numbers
        e. significance_level = the numeric level of significance you want to evaluate p-values by.
            i. Do not use quotes
            ii. If you are unsure, it's a rule of thumb to use 0.05

iii. Examples:
    1. If I want to know if there is a significant difference between past 30 day use for a variety of substances comparing genders (where 1 = Female, and 2 = Male) I would do the following:
        a. ttest_p30d_gender = ttest_2sample(df, ['P30D_Cigs', 'P30D_Vapes', 'P30D_Cigars'], 'Gender', [1,2], 0.05)
            i. This would produce a table called ttest_p30d_gender that contains information on each of the tests ran (which IVs and DVs), test statistics, and a column that identifies if results are significant at the 0.05 level.