proj03

April 26, 2024

1 Import Packages

[ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.formula.api as smf
from statsmodels.stats.multitest import multipletests
from stargazer.stargazer import Stargazer

from proj03 import cgmwildboot

%load_ext autoreload
%autoreload 2
```

2 Import data

[ ]:

```
# import data
baseline = pd.read_stata('data/baseline.dta')
cleanpricedata_y1y2 = pd.read_stata('data/cleanPriceData_Y1Y2.dta')
ms1ms2_pooled = pd.read_stata('data/MS1MS2_pooled.dta')

# this data is not needed for our analysis
# bok_inflation = pd.read_stata('data/BOK_inflation.dta')
# intensity_obs_short = pd.read_stata('data/intensity_obs_short.dta')
# lrfu_select_dataset = pd.read_stata('data/LRFU_select_dataset.dta')
# repayment_datay1 = pd.read_stata('data/repayment_dataY1.dta')
```

3 Recreating the tables from the paper

3.1 Table 1

We start by cleaning the data

[ ]:

```
# clean ms1ms2_pooled (drop if MS !=2, keep columns oafid and treatMS1MS2,␣
 ↪group by oafid and take mean and rename)
ms1ms2_pooled_tab1 = ms1ms2_pooled[ms1ms2_pooled['MS']==2]
```

```
ms1ms2_pooled_tab1 = ms1ms2_pooled_tab1[['oafid', 'treatMS1MS2']]
ms1ms2_pooled_tab1 = ms1ms2_pooled_tab1.groupby('oafid', as_index=False).mean()
ms1ms2_pooled_tab1.rename(columns={'treatMS1MS2': 'treat13'}, inplace=True)
```

[ ]:

```
# clean baseline data (the stata code indicates that the variables columns␣
 ↪'businessprofitmonth' and 'delta' should be kept, however they have already␣
 ↪been renamed to 'businessprofitmonth_base' and 'delta_base')
base_cols = ['oafid', 'logtotcons_base', 'male', 'num_adults',␣
 ↪'num_schoolchildren', 'finished_primary',
                       'finished_secondary', 'cropland', 'num_rooms', 'schoolfees',␣
 ↪'totcons_base', 'logpercapcons_base',
                       'total_cash_savings_base', 'total_cash_savings_trimmed',␣
 ↪'has_savings_acct', 'taken_bank_loan',
                       'taken_informal_loan', 'liquidWealth', 'wagepay',␣
 ↪'businessprofitmonth_base', 'price_avg_diff_pct',
                       'price_expect_diff_pct', 'harvest2011', 'netrevenue2011',␣
 ↪'netseller2011', 'autarkic2011',
                       'maizelostpct2011', 'harvest2012', 'correct_interest',␣
 ↪'digit_recall', 'maizegiver', 'delta_base', 'treatment']
baseline_clean = baseline[base_cols].copy()

# rename columns
baseline_clean.columns = [col + '_base' if not col.endswith('_base') and col !=␣
 ↪'oafid' and col != 'treatment' else col for col in baseline_clean.columns]
baseline_clean.rename(columns={'treatment': 'treatment2012'}, inplace=True)

# generate treat12 as bool for treatment and control in 2012
baseline_clean['treat12'] = baseline_clean['treatment2012'].apply(lambda x: x␣
 ↪in ['T1', 'T2'])
baseline_clean.loc[baseline_clean['treatment2012'] == '', 'treat12'] = np.nan
```

```
/var/folders/yw/jsw5n53s1cb1s2q6tt0msrm00000gn/T/ipykernel_89498/2284489521.py:1
6: FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value 'nan' has dtype incompatible
with bool, please explicitly cast to a compatible dtype first.
  baseline_clean.loc[baseline_clean['treatment2012'] == '', 'treat12'] = np.nan
```

[ ]:

```
# merge baseline_clean and ms1ms2_pooled_clean on oafid
base_ms1ms2_pool = pd.merge(baseline_clean, ms1ms2_pooled_tab1, on='oafid',␣
 ↪how='left')
```

[ ]:

```
# create table 1
# copy in case we need this later
df_tab1 = base_ms1ms2_pool.copy()
df_tab1['schoolfees_base'] = df_tab1['schoolfees_base']*1000

# var list for table 1
```

```
vars_list = [
    "male_base", "num_adults_base", "num_schoolchildren_base",␣
 ↪"finished_primary_base",
    "finished_secondary_base", "cropland_base", "num_rooms_base",␣
 ↪"schoolfees_base",
    "totcons_base", "logpercapcons_base", "total_cash_savings_base",
    "total_cash_savings_trimmed_base", "has_savings_acct_base",␣
 ↪"taken_bank_loan_base",
    "taken_informal_loan_base", "liquidWealth_base", "wagepay_base",
    "businessprofitmonth_base", "price_avg_diff_pct_base",
    "price_expect_diff_pct_base", "harvest2011_base", "netrevenue2011_base",
    "netseller2011_base", "autarkic2011_base", "maizelostpct2011_base",
    "harvest2012_base", "correct_interest_base", "digit_recall_base",
    "maizegiver_base"
]

renaming = {
    "male_base": "Male",
    "num_adults_base": "Number of adults",
    "num_schoolchildren_base": "Children in school",
    "finished_primary_base": "Finished primary school",
    "finished_secondary_base": "Finished secondary school",
    "cropland_base": "Total cropland (acres)",
    "num_rooms_base": "Number of rooms in household",
    "schoolfees_base": "Total school fees",
    "totcons_base": "Average monthly consumption (Ksh)",
    "logpercapcons_base": "Average monthly consumption/capita (log)",
    "total_cash_savings_base": "Total cash savings (Ksh)",
    "total_cash_savings_trimmed_base": "Total cash savings (trim)",
    "has_savings_acct_base": "Has bank savings acct",
    "taken_bank_loan_base": "Taken bank loan",
    "taken_informal_loan_base": "Taken informal loan",
    "liquidWealth_base": "Liquid wealth (Ksh)",
    "wagepay_base": "Off-farm wages (Ksh)",
    "businessprofitmonth_base": "Business profit (Ksh)",
    "price_avg_diff_pct_base": "Avg $\%\Delta$ price Sep-Jun",
    "price_expect_diff_pct_base": "Expect $\%\Delta$ price Sep12-Jun13",
    "harvest2011_base": "2011 LR harvest (bags)",
    "netrevenue2011_base": "Net revenue 2011 (Ksh)",
    "netseller2011_base": "Net seller 2011",
    "autarkic2011_base": "Autarkic 2011",
    "maizelostpct2011_base": "\% maize lost 2011",
    "harvest2012_base": "2012 LR harvest (bags)",
    "correct_interest_base": "Calculated interest correctly",
    "digit_recall_base": "Digit span recall",
    "maizegiver_base": "Maize giver"
}
```

```
# function to perform t-tests
def t_test_by_group(df, var, group_var='treat12'):
    group1 = df[df[group_var] == 0][var].dropna()
    group2 = df[df[group_var] == 1][var].dropna()
    t_stat, p_val = stats.ttest_ind(group1, group2, equal_var=True)
    return group1.mean(), group2.mean(), len(group1) + len(group2), t_stat,␣
 ↪p_val

# applying t-tests and collecting results
results = []
for var in vars_list:
    control_mean, treat_mean, obs, t_stat, p_val = t_test_by_group(df_tab1, var)
    std_diff = (treat_mean - control_mean) / np.std(df_tab1[df_tab1['treat12']␣
 ↪== 0][var])
    results.append([var, treat_mean, control_mean, obs, std_diff, p_val])

# convert results to a df to use pandas output to latex
results_df = pd.DataFrame(results, columns=['Variable', 'Treat Mean', 'Control␣
 ↪Mean', 'Observations', 'Std Diff', 'P-value'])
results_df['Variable'] = results_df['Variable'].map(renaming)
results_df = results_df.rename(columns={
    'Variable':'Baseline characteristic',
    'Treat Mean':'Treat',
    'Control Mean':'Control',
    'Observations':'Obs',
    'Std Diff':'Std diff',
    'P-value':'P-val'})

latex_table1 = results_df.to_latex(index=False, float_format="%.3f")
latex_table1 = latex_table1.replace('\\toprule', '\\\[-1.8ex]\hline \n \hline␣
 ↪\\\[-1.8ex]')
latex_table1 = latex_table1.replace('\\bottomrule', '\\\[-1.8ex]\hline \n␣
 ↪\hline \\\[-1.8ex]')

with open('tables/table1.tex','w') as file:
    file.write(latex_table1)
```

3.2 Running the model for tables 2 through 4

[ ]:

```
treatments = ['treat12', 'treat13', 'treatMS1MS2']
dependent_vars = ['inventory_trim', 'netrevenue_trim','logtotcons_trim']

mean_df = pd.DataFrame()
std_df = pd.DataFrame()
pval_df = pd.DataFrame()
pval_rd_df = pd.DataFrame()
```

```
results = {'netsales': {'overall': None, 'by_round':None}}

for dv in dependent_vars:
    for treat in treatments:
        # create df for each treatment
        if treat == 'treatMS1MS2':
            df1 = ms1ms2_pooled.loc[:, [dv,'treat12', 'Y1round1', 'Y1round2',␣
 ↪'Y1round3', 'treatMS1MS2', 'interviewdate', 'groupnum', 'strata_group']].
 ↪copy(deep=True).dropna()
            df2 = ms1ms2_pooled.loc[:, [dv,'treat13', 'Y2round1', 'Y2round2',␣
 ↪'Y2round3', 'treatMS1MS2', 'interviewdate', 'groupnum', 'strata_group']].
 ↪copy(deep=True).dropna()
            df1['inter_R1'] = df1['Y1round1'] * df1[f'treat12']
            df1['inter_R2'] = df1['Y1round2'] * df1[f'treat12']
            df1['inter_R3'] = df1['Y1round3'] * df1[f'treat12']
            df2['inter_R1'] = df2['Y2round1'] * df2[f'treat13']
            df2['inter_R2'] = df2['Y2round2'] * df2[f'treat13']
            df2['inter_R3'] = df2['Y2round3'] * df2[f'treat13']
            df = pd.concat([df1, df2], ignore_index=True).fillna(0)

            # model specification by round
            formula_by_round = f'{dv} ~ inter_R1 + inter_R2 + inter_R3 +␣
 ↪interviewdate + C(Y1round1) + C(Y1round2) + C(Y1round3) + C(Y2round1) +␣
 ↪C(Y2round2) + C(Y2round3) + C(strata_group)'
        else:
            if treat == 'treat12':
                year = 1
            else:
                year = 2
            df = ms1ms2_pooled.loc[:, [dv,treat, f'Y{year}round1',␣
 ↪f'Y{year}round2', f'Y{year}round3', 'treatMS1MS2', 'interviewdate',␣
 ↪'groupnum', 'strata_group']].copy(deep=True).dropna()
            df['inter_R1'] = df[f'Y{year}round1'] * df[f'{treat}']
            df['inter_R2'] = df[f'Y{year}round2'] * df[f'{treat}']
            df['inter_R3'] = df[f'Y{year}round3'] * df[f'{treat}']


            # model specification by round
            formula_by_round = f'{dv} ~ inter_R1 + inter_R2 + inter_R3 +␣
 ↪interviewdate + C(Y{year}round1) + C(Y{year}round2) + C(Y{year}round3) +␣
 ↪C(strata_group)'

        df['z'] = df[treat]

        # specify overall model
        formula_overall = f'{dv} ~ z + interviewdate + C(strata_group)'
        # fit models
        model_overall = smf.ols(formula_overall, data=df).
 ↪fit(cov_type='cluster', cov_kwds={'groups': df['groupnum']})
        model_by_round = smf.ols(formula_by_round, data=df).
 ↪fit(cov_type='cluster', cov_kwds={'groups': df['groupnum']})

        # store models in dictionary
        results[f'{treat}_{dv}'] = {f'overall': model_overall, f'by_round':␣
 ↪model_by_round}
        # extract necessary statistics
        mean_df.loc[dv, treat] = df[dv].mean()
        std_df.loc[dv, treat] = df[dv].std()
        pval_df.loc[dv, treat] = 2 * (1 - stats.t.cdf(np.abs(model_overall.
 ↪params['z']/model_overall.bse['z']),df=df['groupnum'].nunique()-1))

        for var in ['inter_R1', 'inter_R2', 'inter_R3']:
            pval_rd_df.loc[f'{dv}_{var}', f'{treat}_rd'] = 2 * (1 - stats.t.
 ↪cdf(np.abs(model_by_round.params[var]/model_by_round.
 ↪bse[var]),df=df['groupnum'].nunique()-1))
```

3.3 Adding table 5

3.3.1 Clean the data

[ ]:

```
ms1ms2_pooled_tab5 = ms1ms2_pooled.copy(deep=True)
max_strata_group = ms1ms2_pooled_tab5['strata_group'].max()
ms1ms2_pooled_tab5.loc[ms1ms2_pooled_tab5['MS'] == 2, 'strata_group'] =␣
 ↪ms1ms2_pooled_tab5['groupstrata'] + max_strata_group

ms1ms2_pooled_tab5.loc[ms1ms2_pooled_tab5['MS'] == 2, 'oafid'] =␣
 ↪ms1ms2_pooled_tab5['fr_id']

ms1ms2_pooled_tab5['purchasequant2'] = ms1ms2_pooled_tab5['purchasequant']
ms1ms2_pooled_tab5.
 ↪loc[(ms1ms2_pooled_tab5['purchaseval']==0)&(ms1ms2_pooled_tab5['purchasequant'].
 ↪isna()),'purchasequant2'] = 0
ms1ms2_pooled_tab5['netsales'] = ms1ms2_pooled_tab5['salesquant'] -␣
 ↪ms1ms2_pooled_tab5['purchasequant2']

ms1ms2_pooled_tab5.
 ↪drop(columns=['netsales_trim','purchaseval_trim','salesval_trim'],␣
 ↪inplace=True)
```

[ ]:

```
# trim outliers
for x in ['purchaseval', 'salesval', 'purchasequant', 'salesquant']:
    quantile = np.quantile(ms1ms2_pooled_tab5[ms1ms2_pooled_tab5[x].
  ↪notna()][x],[0.99],method='closest_observation')
    ms1ms2_pooled_tab5[f'{x}_trim'] = ms1ms2_pooled_tab5[x]
    ms1ms2_pooled_tab5.loc[ms1ms2_pooled_tab5[f'{x}_trim'] >␣
  ↪quantile[0],f'{x}_trim'] = np.nan

quantile = np.quantile(ms1ms2_pooled_tab5[ms1ms2_pooled_tab5['netsales'].
  ↪notna()]['netsales'],[0.005, 0.995],method='closest_observation')
ms1ms2_pooled_tab5['netsales_trim'] = ms1ms2_pooled_tab5['netsales']
ms1ms2_pooled_tab5.loc[(ms1ms2_pooled_tab5['netsales_trim'] <= quantile[0]) |␣
  ↪(ms1ms2_pooled_tab5['netsales_trim'] > quantile[1]) , 'netsales_trim'] = np.
  ↪nan

# create id
ms1ms2_pooled_tab5['id'] = ms1ms2_pooled_tab5['oafid'].
  ↪fillna(ms1ms2_pooled_tab5['fr_id'])

# create effective prices
trim_vars = ['salesquant_trim', 'purchasequant_trim', 'salesval_trim',␣
  ↪'purchaseval_trim']
for var in trim_vars:
    ms1ms2_pooled_tab5[f'tot_{var}'] = ms1ms2_pooled_tab5.groupby(['id',␣
  ↪'MS'])[var].transform('sum')

for x in ['purchase', 'sales']:
    ms1ms2_pooled_tab5[f'effective_{x}_price'] =␣
  ↪ms1ms2_pooled_tab5[f'tot_{x}val_trim'] /␣
  ↪ms1ms2_pooled_tab5[f'tot_{x}quant_trim']
    ms1ms2_pooled_tab5.loc[ms1ms2_pooled_tab5[f'tot_{x}quant_trim']==␣
  ↪0,f'effective_{x}_price'] = np.nan
```

3.3.2 Net sales

```
[ ]: # define variable
dv = 'netsales_trim'
independent_vars = ['z', 'treatMS1MS2_1 + treatMS1MS2_2 + treatMS1MS2_3']

for i, var in enumerate(independent_vars):
    df = ms1ms2_pooled_tab5.copy(deep=True)
    df['z'] = df['treatMS1MS2']
    if var == 'z':
        df.
  ↪dropna(subset=[dv,'z','interviewdate','Y1round2','Y1round3','Y2round1','Y2round2','Y2round3
  ↪inplace=True)
    else:
        df.
↪dropna(subset=[dv,'treatMS1MS2_1','treatMS1MS2_2','treatMS1MS2_3','interviewdate','Y1round2
↪inplace=True)
    df.reset_index(drop=True, inplace=True)

    formula = f'{dv} ~ {var} + interviewdate + Y1round2 + Y1round3 + Y2round1 +␣
↪Y2round2 + Y2round3 + C(strata_group)'
    model = smf.ols(formula, df).fit(cov_type='cluster', cov_kwds={'groups':␣
↪df['groupnum']})
    if i == 0:
        results['netsales']['overall'] = model
    else:
        results['netsales']['by_round'] = model

    mean_df.loc[dv, treat] = df.loc[df['treatMS1MS2'] == 0, dv].mean()
    std_df.loc[dv, treat] = df.loc[df['treatMS1MS2'] == 0, dv].std()
```

```
/var/folders/yw/jsw5n53s1cb1s2q6tt0msrm00000gn/T/ipykernel_89498/2854517969.py:2
1: FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value '-0.4208270957978571' has
dtype incompatible with float32, please explicitly cast to a compatible dtype
first.
  mean_df.loc[dv, treat] = df.loc[df['treatMS1MS2'] == 0, dv].mean()
```

3.3.3 Effective Price

```
[ ]: for dv in ['purchase', 'sales']:
         for i, treat in enumerate(['treat12', 'treat13', 'treatMS1MS2']):
             df = ms1ms2_pooled_tab5.copy(deep=True)
             df['z'] = df[treat]
             df = df.drop_duplicates(subset=['id', 'MS'], keep='first')
             df.dropna(subset=[f'effective_{dv}_price','z','groupnum'], inplace=True)
             if treat == 'treatMS1MS2':
                 formula = f'effective_{dv}_price ~ z + C(strata_group)'
             else:
                 df = df[df['MS'] == i+1]
                 formula = f'effective_{dv}_price ~ z + C(strata_group)'
             model = smf.ols(formula, data=df).fit(cov_type='cluster',␣
     ↪cov_kwds={'groups': df['groupnum']})
             results[f'{treat}_{dv}'] = {'overall':model}

             mean_df.loc[dv, treat] = df.loc[df['z'] == 0, f'effective_{dv}_price'].
     ↪mean()
             std_df.loc[dv, treat] = df.loc[df['z'] == 0, f'effective_{dv}_price'].
     ↪std()
             pval_df.loc[dv, treat] = 2 * (1 - stats.t.cdf(np.abs(model.params['z']/
     ↪model.bse['z']),df=df['groupnum'].nunique()-1))
```

```
/var/folders/yw/jsw5n53s1cb1s2q6tt0msrm00000gn/T/ipykernel_89498/1060375524.py:1
5: FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value '2774.7609839746265' has
dtype incompatible with float32, please explicitly cast to a compatible dtype
first.
  mean_df.loc[dv, treat] = df.loc[df['z'] == 0, f'effective_{dv}_price'].mean()
/var/folders/yw/jsw5n53s1cb1s2q6tt0msrm00000gn/T/ipykernel_89498/1060375524.py:1
5: FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value '2858.969741383102' has
dtype incompatible with float32, please explicitly cast to a compatible dtype
first.
  mean_df.loc[dv, treat] = df.loc[df['z'] == 0, f'effective_{dv}_price'].mean()
```

3.4 Calculating FWER and pvals and getting dataframes ready for output

[ ]:

```
for treat in ['treat12', 'treat13', 'treatMS1MS2']:
    fwer_pvals = multipletests(pval_df[treat], alpha=0.05, method='fdr_bh')[1]
    for i, dv in enumerate(pval_df.index):
        pval_df.loc[dv, f'{treat}_fwer'] = fwer_pvals[i]
    fwer_pvals_rd = multipletests(pval_rd_df[f'{treat}_rd'], alpha=0.05,␣
  ↪method='fdr_bh')[1]
    for i, indx in enumerate(pval_rd_df.index):
        pval_rd_df.loc[indx, f'{treat}_fwer_rd'] = fwer_pvals_rd[i]
```

[ ]:

```
# combine the p-values and split into two dfs
pvals = pd.concat([pval_df, pval_rd_df], axis=0)
pvals = pvals.map(lambda x: '<0.001' if x < 0.0005 else np.round(x,3))
pval_fwer =␣
  ↪pvals[['treat12_fwer','treat12_fwer_rd','treat13_fwer','treat13_fwer_rd','treatMS1MS2_fwer'
pval =␣
  ↪pvals[['treat12','treat12_rd','treat13','treat13_rd','treatMS1MS2','treatMS1MS2_rd']]
```

[ ]:

```
# adjust the mean and std dfs to be ready for output
for treat in mean_df.columns:
    mean_df[f'{treat}_rd'] = mean_df[treat]
    std_df[f'{treat}_rd'] = std_df[treat]

# sort the dfs
mean_df =␣
  ↪mean_df[['treat12','treat12_rd','treat13','treat13_rd','treatMS1MS2','treatMS1MS2_rd']].
  ↪map(lambda x: np.round(x,3))
std_df =␣
  ↪std_df[['treat12','treat12_rd','treat13','treat13_rd','treatMS1MS2','treatMS1MS2_rd']].
  ↪map(lambda x: np.round(x,3))
```

3.5 Output table to LaTeX

3.5.1 Tables 2,3 and 4 to LaTeX

```
[ ]: latex_tables = []
     for i, dv in enumerate(['inventory_trim', 'netrevenue_trim',␣
      ↪'logtotcons_trim']):
         tables = []
         for treat in ['treat12', 'treat13', 'treatMS1MS2']:
             overall = results[f'{treat}_{dv}']['overall']
             by_rd = results[f'{treat}_{dv}']['by_round']
             tables.append(overall)
             tables.append(by_rd)
         stargazer = Stargazer(tables)
         stargazer.custom_columns(['Y1', 'Y2','Pooled'], [2,2,2])
         stargazer.significant_digits(3)
         stargazer.rename_covariates({'z': 'Treat', 'inter_R1': 'Treat - R1',␣
      ↪'inter_R2': 'Treat - R2', 'inter_R3': 'Treat - R3'})
         stargazer.covariate_order(['z', 'inter_R1', 'inter_R2', 'inter_R3'])
         stargazer.show_adj_r2 = False
         stargazer.show_f_statistic = False
         stargazer.show_residual_std_err = False
         stargazer.show_notes = False

         # adding custom rows with mean, sd, and p-values
         stargazer.add_line('Mean DV', mean_df.loc[dv].tolist(),location='fb')
         stargazer.add_line('SD DV', std_df.loc[dv].tolist(),location='fb')
         stargazer.add_line('P-Val Treat', pval.loc[dv].tolist(),location='fb')
         stargazer.add_line('P-Val Treat FWER', pval_fwer.loc[dv].
      ↪tolist(),location='fb')
         stargazer.add_line('P-Val Treat - R1', pval.loc[f'{dv}_inter_R1'].
      ↪tolist(),location='fb')
         stargazer.add_line('P-Val Treat - R1 FWER', pval_fwer.loc[f'{dv}_inter_R1'].
      ↪tolist(),location='fb')
         stargazer.add_line('P-Val Treat - R2', pval.loc[f'{dv}_inter_R2'].
      ↪tolist(),location='fb')
         stargazer.add_line('P-Val Treat - R2 FWER', pval_fwer.loc[f'{dv}_inter_R2'].
      ↪tolist(),location='fb')
         stargazer.add_line('P-Val Treat - R3', pval.loc[f'{dv}_inter_R3'].
      ↪tolist(),location='fb')
         stargazer.add_line('P-Val Treat - R3 FWER', pval_fwer.loc[f'{dv}_inter_R3'].
      ↪tolist(),location='fb')

         latex_table = stargazer.render_latex()

         # general formatting
         latex_table = latex_table.replace("\\textit{Note","% \\textit{Note")
```

```python
    latex_table = latex_table.replace("nan","")
    latex_table = latex_table.replace("\\begin{table}[!htbp] \\centering", "")
    latex_table = latex_table.replace("\\end{table}", "")

    # renaming variables
    latex_table = latex_table.replace("\\[-1.8ex] & (1) & (2) & (3) & (4) & (5)␣
↪& (6) \\",
                                "\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) & (6)␣
↪\n \\\ & Overall & By rd & Overall & By rd & Overall & By rd \\")
    latex_table = latex_table.replace("netrevenue_trim","Net Revenue Trim")
    latex_table = latex_table.replace("inventory_trim","Inventory Trim")
    latex_table = latex_table.replace("logtotcons_trim","Log Total HH␣
↪Consumption Trim")

    latex_tables.append(latex_table)
```

```
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 37, but rank is 36
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 37, but rank is 36
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 63, but rank is 62
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 71, but rank is 68
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 37, but rank is 36
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 37, but rank is 36
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 63, but rank is 62
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 71, but rank is 68
```

```
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 37, but rank is 36
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 37, but rank is 36
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 63, but rank is 62
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 71, but rank is 68
  warnings.warn('covariance of constraints does not have full '
```

3.5.2 Saving Tables

[ ]:

```
# write to file
with open('tables/table2.tex','w') as file:
    file.write(latex_tables[0])

with open('tables/table3.tex','w') as file:
    file.write(latex_tables[1])

with open('tables/table4.tex','w') as file:
    file.write(latex_tables[2])
```

3.5.3 Table 5 to LaTeX

[ ]:

```
tables = [results['netsales']['overall'],␣
 ↪results['netsales']['by_round'],results['treatMS1MS2_purchase']['overall'],␣
 ↪results['treatMS1MS2_sales']['overall']]

stargazer = Stargazer(tables)
stargazer.custom_columns(['Net Sales', 'Effective Price'], [2, 2])
stargazer.rename_covariates({'z': 'Treat','treatMS1MS2_1':'Treat - R1',␣
 ↪'treatMS1MS2_2':'Treat - R2', 'treatMS1MS2_3':'Treat - R3'})
stargazer.significant_digits(3)
stargazer.covariate_order(['z', 'treatMS1MS2_1', 'treatMS1MS2_2',␣
 ↪'treatMS1MS2_3'])
stargazer.show_adj_r2 = False
stargazer.show_f_statistic = False
stargazer.show_residual_std_err = False
stargazer.show_notes = False
```

```python
# adding p-values
stargazer.add_line('Mean DV', mean_df.
 ↪loc[['netsales_trim','netsales_trim','purchase','sales'],'treatMS1MS2'].
 ↪tolist(),location='fb')
stargazer.add_line('SD DV', std_df.
 ↪loc[['netsales_trim','netsales_trim','purchase','sales'],'treatMS1MS2'].
 ↪tolist(),location='fb')
stargazer.add_line('P-Val Treat', ['','']+pval.
 ↪loc[['purchase','sales'],'treatMS1MS2'].tolist(),location='fb')
stargazer.add_line('P-Val Treat FWER', ['','']+pval_fwer.
 ↪loc[['purchase','sales'],'treatMS1MS2_fwer'].tolist(),location='fb')

latex_table5 = stargazer.render_latex()

# general formatting
latex_table5 = latex_table5.replace("nan","")
latex_table5 = latex_table5.replace("\\begin{table}[!htbp] \\centering", "")
latex_table5 = latex_table5.replace("\\end{table}", "")

# renaming variables
latex_table5 = latex_table5.replace("\\[-1.8ex] & (1) & (2) & (3) & (4) \\",
                                "\\[-1.8ex] & (1) & (2) & (3) & (4) \n \\\ &␣
 ↪Overall & By rd & Purchase & Sales \\")


with open('tables/table5.tex','w') as file:
    file.write(latex_table5)
```

```
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 68, but rank is 66
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 70, but rank is 68
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 62, but rank is 61
  warnings.warn('covariance of constraints does not have full '
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/base/model.py:1888:
ValueWarning: covariance of constraints does not have full rank. The number of
constraints is 62, but rank is 61
  warnings.warn('covariance of constraints does not have full '
```

3.6 Table 6

3.7 Clean data

[ ]:
```
cleanpricedata_y1y2_tab6 = cleanpricedata_y1y2.copy(deep=True)
cleanpricedata_y1y2_tab6 =␣
 ↪cleanpricedata_y1y2_tab6[['salesPrice_trim','hi_1km_wt','hi_3km_wt','hi_5km_wt','monthnum',
 ↪'in_sample','MS','lean']]
cleanpricedata_y1y2_tab6['hi'] = pd.NA
cleanpricedata_y1y2_tab6['interact'] = pd.NA
cleanpricedata_y1y2_tab6['interact_lean'] = pd.NA
```

3.8 Run first set of regressions

[ ]:
```
results = {}
for dist in ['1km_wt', '3km_wt', '5km_wt']:
    df = cleanpricedata_y1y2_tab6.copy(deep=True)
    df.dropna(subset=[f'hi_{dist}','salesPrice_trim','monthnum'], inplace=True)
    mean_price = df[(df['monthnum'] == 0) & (df[f'hi_{dist}'] ==␣
 ↪0)]['salesPrice_trim'].mean()
    norm = 100 / mean_price

    # normalize price
    df['salesPrice_trim_norm'] = df['salesPrice_trim'] * norm

    # create hi variable
    df['hi'] = df[f'hi_{dist}']
    df['interact'] = df['monthnum'] * df['hi']

    # regression
    formula = 'salesPrice_trim_norm ~ hi + monthnum + interact'

    for ms in [1,2,3]: # 3 is pooled
        if ms == 3:
            df_filt = df[(df['in_sample'] == 1)]
        else:
            df_filt = df[(df['MS'] == ms) & (df['in_sample'] == 1)]
        model = smf.ols(formula=formula, data=df_filt).fit(cov_type='cluster',␣
 ↪cov_kwds={'groups': df_filt[f'subloc_{dist}_grp']})
        results[(dist, ms)] = model
```

[ ]:
```
pvals = pd.DataFrame()
# calculating the adjusted p-values using the t-statistic with cluster-1␣
 ↪degrees of freedom
for dv in ['hi', 'monthnum', 'interact']:
    pval = {(k[0], k[1]): 2 * (1 - stats.t.cdf(abs(v.params[dv] / v.
 ↪bse[dv]),df=cleanpricedata_y1y2_tab6[f'subloc_{k[0]}_grp'].nunique()-1)) for␣
 ↪k, v in results.items()}
```

```
    pvals[dv] = pd.Series(pval)
```

3.9 Run bootstrap iterations

[ ]:

```
n_bootstraps = 5000 # reported data is based on 5000 iterations
bootstrap_ests = {}
bootstrap_pvals = pd.DataFrame(index=pd.MultiIndex.from_product([['1km_wt',␣
 ↪'3km_wt', '5km_wt'], [1, 2, 3]], names=['dist', 'ms']), columns=['hi',␣
 ↪'monthnum', 'interact'])
bootstrap_pvals_test = pd.DataFrame(index=pd.MultiIndex.
 ↪from_product([['1km_wt', '3km_wt', '5km_wt'], [1, 2, 3]], names=['dist',␣
 ↪'ms']), columns=['hi', 'monthnum', 'interact'])

for dist  in ['1km_wt', '3km_wt', '5km_wt']:
    df = cleanpricedata_y1y2_tab6.copy(deep=True)
    df.dropna(subset=[f'hi_{dist}','salesPrice_trim','monthnum'], inplace=True)
    mean_price = df[(df['monthnum'] == 0) & (df[f'hi_{dist}'] ==␣
 ↪0)]['salesPrice_trim'].mean()
    norm = 100 / mean_price

    # normalize price
    df['salesPrice_trim_norm'] = df['salesPrice_trim'] * norm
    df['salesPrice_trim_norm'] = df['salesPrice_trim_norm'].astype(float)

    # create hi variable
    df['hi'] = df[f'hi_{dist}']
    df['interact'] = df['monthnum'] * df['hi']

    # regression
    formula = 'salesPrice_trim_norm ~ hi + monthnum + interact'

    for ms in [1,2,3]: # 3 is pooled
        if ms == 3:
            df_filt = df[(df['in_sample'] == 1)]
        else:
            df_filt = df[(df['MS'] == ms) & (df['in_sample'] == 1)]

        model = results[(dist, ms)]

        boot_ests, boot_pval = cgmwildboot(df_filt, model,n_bootstraps,␣
 ↪f'subloc_{dist}_grp',f'subloc_{dist}_grp',seed=5005)
        bootstrap_ests[(dist,ms)] = boot_ests
        bootstrap_pvals.loc[(dist,ms)] = boot_pval
```

3.10 Adjusting pval tables

```
[ ]: # keep only columns 3km_wt and 3rd column in 1km_wt and 5km_wt
     pvals = pvals.T
     pvals = pvals[[('3km_wt', 1), ('3km_wt', 2), ('3km_wt', 3), ('1km_wt', 3),␣
      ↪('5km_wt', 3)]]
     pvals = pvals.map(lambda x: '<0.001' if x < 0.0005 else np.round(x,3))

     bootstrap_pvals = bootstrap_pvals.T
     bootstrap_pvals = bootstrap_pvals[[('3km_wt', 1), ('3km_wt', 2), ('3km_wt', 3),␣
      ↪('1km_wt', 3), ('5km_wt', 3)]]
     bootstrap_pvals = bootstrap_pvals.map(lambda x: '<0.001' if x < 0.0005 else np.
      ↪round(x,3))
```

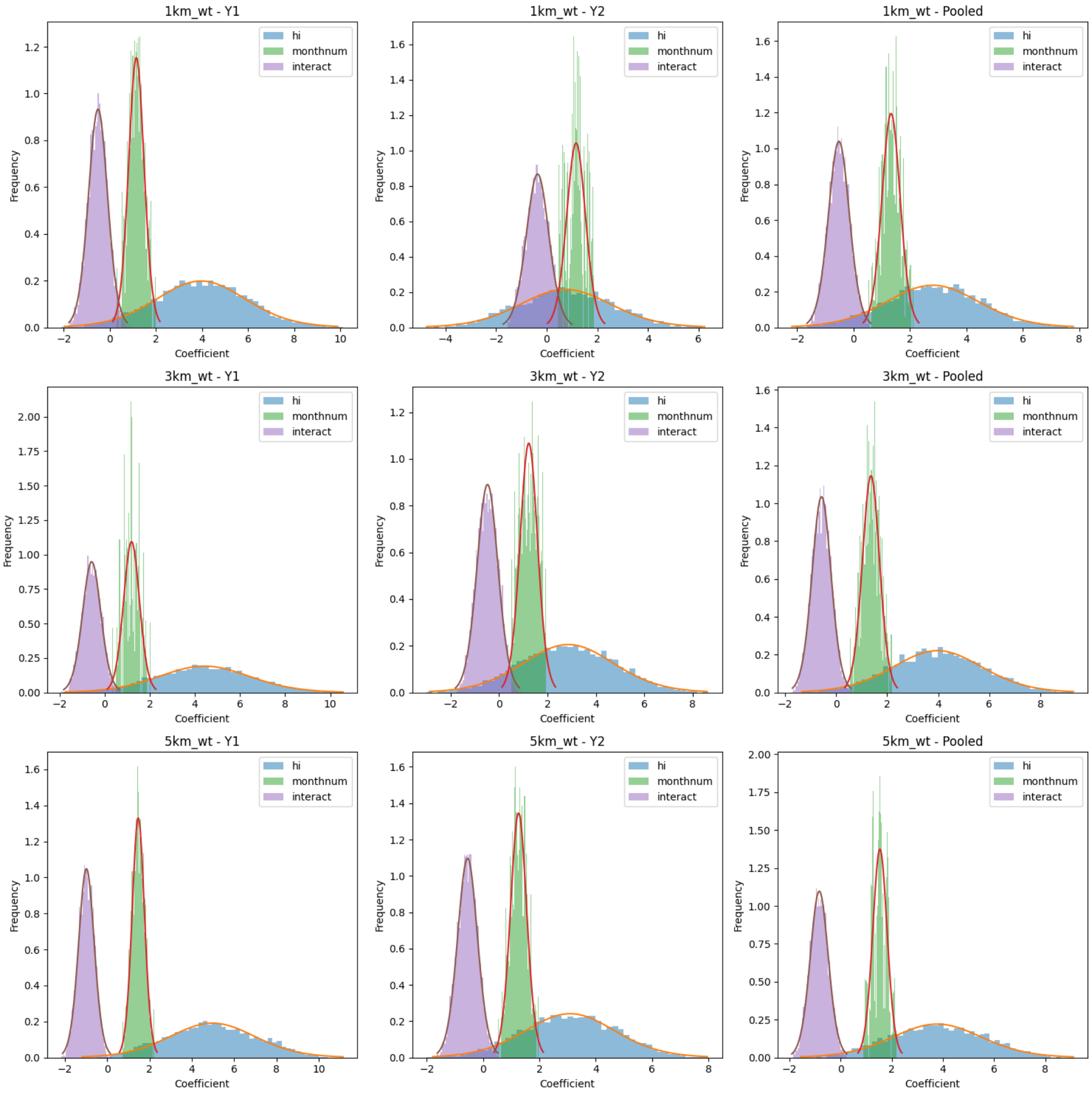
3.11 Ouput to LaTeX

```
[ ]: # use stargazer to create a table
     result_list = [results[('3km_wt', 1)], results[('3km_wt', 2)],␣
      ↪results[('3km_wt', 3)], results[('1km_wt', 3)], results[('5km_wt', 3)]]
     stargazer = Stargazer(result_list)

     # configure Stargazer object for output
     stargazer.custom_columns(['Main Specification (3km)', 'Robustness (Pooled)'],␣
      ↪[3, 2])
     stargazer.rename_covariates({'hi': 'High', 'monthnum': 'Month', 'interact':␣
      ↪'High x Month'})
     stargazer.show_degrees_of_freedom(False)
     stargazer.significant_digits(3)
     stargazer.covariate_order(['hi', 'monthnum', 'interact'])
     stargazer.show_adj_r2 = False
     stargazer.show_f_statistic = False
     stargazer.show_residual_std_err = False
     stargazer.show_notes = False

     # adding custom rows with p-values
     stargazer.add_line('P-value High', pvals.loc['hi'].values.
      ↪tolist(),location='fb')
     stargazer.add_line('P-value High Bootstrap', bootstrap_pvals.loc['hi'].values.
      ↪tolist(),location='fb')
     stargazer.add_line('P-value Month', pvals.loc['monthnum'].values.
      ↪tolist(),location='fb')
     stargazer.add_line('P-value High Bootstrap', bootstrap_pvals.loc['monthnum'].
      ↪values.tolist(),location='fb')
     stargazer.add_line('P-value High x Month', pvals.loc['interact'].values.
      ↪tolist(),location='fb')
     stargazer.add_line('P-value High x Month Bootstrap', bootstrap_pvals.
      ↪loc['interact'].values.tolist(),location='fb')
```

```
latex_table6 = stargazer.render_latex()

# edit the latex tables
latex_table6 = latex_table6.replace("\\[-1.8ex] & (1) & (2) & (3) & (4) & (5)␣
 ↪\\",
                                     "\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) \n␣
 ↪\\\ & Y1 & Y2 & Pooled & 1km & 5km \\")
latex_table6 = latex_table6.replace("salesPrice_trim_norm","Sales Price Trim")
latex_table6 = latex_table6.replace("\\begin{table}[!htbp] \\centering", "")
latex_table6 = latex_table6.replace("\\end{table}", "")

with open('tables/table6.tex','w') as file:
    file.write(latex_table6)
```

3.11.1 Create Appendix figure

```
[ ]: # plot distribution of bootstrapped coefficients
fig, axs = plt.subplots(3, 3, figsize=(15, 15))
for i, dist in enumerate(['1km_wt', '3km_wt', '5km_wt']):
    for j, ms in enumerate([1, 2, 3]):
        for k, var in enumerate(['hi', 'monthnum', 'interact']):
            coef = bootstrap_ests[(dist, ms)][:, k]
            mu = np.mean(coef)
            sigma = np.std(coef)
            x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
            axs[i, j].hist(coef, bins=50, alpha=0.5, label=var, density=True)
            axs[i, j].plot(x, stats.t.pdf(x, df=16, loc=mu, scale=sigma))
            if ms == 3:
                axs[i, j].set_title(f'{dist} - Pooled')
            else:
                axs[i, j].set_title(f'{dist} - Y{ms}')
            axs[i, j].set_xlabel('Coefficient')
            axs[i, j].set_ylabel('Frequency')
            axs[i, j].legend()

plt.tight_layout()
plt.savefig('figures/boot_dist_tab6.png')
```



3.12 Table 7

```
[ ]: # copy the raw data and create columns for treatment and interaction variable
     ms1ms2_pooled_tab7 = ms1ms2_pooled.copy(deep=True)
     # filter relevant columns
     ms1ms2_pooled_tab7 = ms1ms2_pooled_tab7[['oafid', # id
                                               'treat12', 'treat13', 'treatMS1MS2', #␣
       ↪treatment variables
                                               'inventory_trim', 'netrevenue_trim',␣
       ↪'logtotcons_trim', # outcome variables
                                               'Y1round2', 'Y1round3', 'Y2round1',␣
       ↪'Y2round2', 'Y2round3','hi','subloc','interviewdate']] # independent␣
       ↪variables
```

```
ms1ms2_pooled_tab7.sort_index(inplace=True)
ms1ms2_pooled_tab7['z'] = pd.NA
ms1ms2_pooled_tab7['z_hi'] = pd.NA
```

3.12.1 Running the first set of regressions

[ ]:

```
# list of treaments
treatments = ['treat12', 'treat13', 'treatMS1MS2']

# list of dependent variables
dependent_vars = ['inventory_trim', 'netrevenue_trim', 'logtotcons_trim']

# empty dataframes to store mean and std for output
mean_std_df = pd.DataFrame(index=pd.MultiIndex.
 ↪from_product([dependent_vars,treatments], names=['dv','treat']),␣
 ↪columns=['mean','std'])

# list of changeing independent variables depending on the treatment
independent_vars = {
    'treat12': 'Y1round2 + Y1round3',
    'treat13': 'Y2round2 + Y2round3',
    'treatMS1MS2': 'Y1round2 + Y1round3 + Y2round1 + Y2round2 + Y2round3'
    }

# empty dictionary to store results
results = {}
pvals = {var: [] for var in ['z', 'hi', 'z_hi','z+z_hi']}

# Simulating the loop to replace variables and run regressions
for dv in dependent_vars:
    for treat in treatments:
        # Stata automatically omits the missing values in the regression - here␣
 ↪we have to do it manually so we copy the data and drop variables
        df = ms1ms2_pooled_tab7.copy(deep=True)
        df = df.dropna(subset=[dv, treat, 'hi', 'subloc','interviewdate'])

        # store mean and std for output
        mean_std_df.loc[(dv, treat),'mean'] = df.loc[df[treat] == 0, dv].mean()
        mean_std_df.loc[(dv, treat),'std'] = df.loc[df[treat] == 0, dv].std()

        # setting treament variable
        df['z'] = df[treat] # setting z to the treatment variable

        # setting interaction variable
        df['z_hi'] = df[treat]*df['hi'] # setting z_hi to the interaction of␣
 ↪the treatment hi saturation
```

```
        # setting the formula to run the regression
        formula = f'{dv} ~ z + hi + z_hi + interviewdate +␣
↪{independent_vars[treat]}'

        # Run the regression
        model_key = f'model_{dependent_vars.index(dv)*len(treatments) +␣
↪treatments.index(treat)}'
        results[model_key] = smf.ols(formula, data=df).fit(cov_type='cluster',␣
↪cov_kwds={'groups': df['subloc']})

        # test the hypothesis that z + z_hi = 0
        hypothesis = 'z + z_hi = 0'
        t_test = results[model_key].t_test(hypothesis, use_t=True)

        # store p-value round to 3 decimals
        pvals['z+z_hi'].append(t_test.pvalue)

        # calculate t-test p-values for z, hi, z_hi
        for var in ['z', 'hi', 'z_hi']:
            pval = 2 * (1 - stats.t.cdf(abs(results[model_key].params[var] /␣
↪results[model_key].bse[var]),df=df[f'subloc'].nunique()-1))
            pvals[var].append(pval)
```

[ ]:

```
pvals = pd.DataFrame(pvals).T
pvals = pvals.map(lambda x: '<0.001' if x < 0.0005 else np.round(x,3))

mean_std_df['mean'] = mean_std_df['mean'].astype(float).round(3)
mean_std_df['std'] = mean_std_df['std'].astype(float).round(3)
mean_std_df = mean_std_df.T
```

3.12.2 Running boostrap regressions

[ ]:

```
n_bootstraps = 5000  # reported data is based on 5000 iterations
bootstrap_ests = {}
bootstrap_pvals = pd.DataFrame(index=pd.MultiIndex.
↪from_product([dependent_vars, treatments], names=['treatment', 'dep_var']),␣
↪columns=['z','hi','z_hi'])

for dv in dependent_vars:
    for treat in treatments:
        df = ms1ms2_pooled_tab7.copy(deep=True)
        df = df.dropna(subset=[dv, treat, 'hi', 'interviewdate','subloc'])
        df['z'] = df[treat]
        df['z_hi'] = df[treat] * df['hi']
        df[dv] = df[dv].astype(float)
        formula = f'{dv} ~ z + hi + z_hi + interviewdate +␣
 ↪{independent_vars[treat]}'
        model_key = f'model_{dependent_vars.index(dv)*len(treatments) +␣
 ↪treatments.index(treat)}'
        model = results[model_key]

        # Wild bootstrap
        boot_ests, boot_pval = cgmwildboot(df, model,n_bootstraps,␣
 ↪'subloc','subloc',seed=5005)
        bootstrap_ests[(dv,treat)] = boot_ests

        for i, var in enumerate(['z', 'hi', 'z_hi']):
            bootstrap_pvals.loc[(dv,treat),var] = boot_pval[i]
```

[ ]:

```
bootstrap_pvals = bootstrap_pvals.T
bootstrap_pvals = bootstrap_pvals.map(lambda x: '<0.001' if x < 0.0005 else np.
 ↪round(x,3))
```

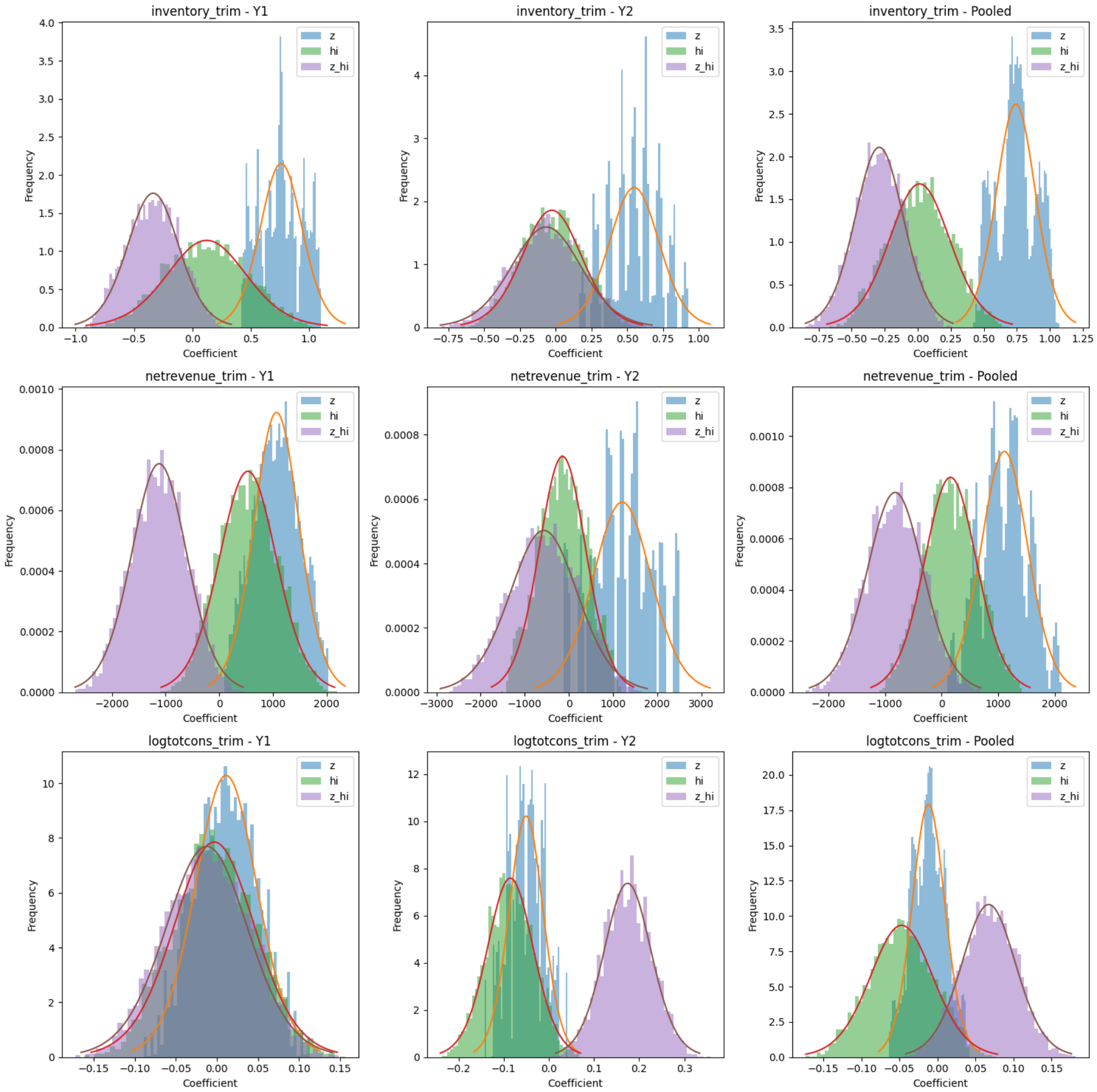
3.12.3 Output to LaTeX

[ ]:

```
# use stargazer to create a table
result_list = list(results.values())
stargazer = Stargazer(result_list)

# configure Stargazer object for output
stargazer.custom_columns(['Inventory', 'Net Revenues', 'Consumption'], [3, 3,␣
 ↪3])
stargazer.rename_covariates({'z': 'Treat', 'hi': 'High', 'z_hi': 'Treat x␣
 ↪High'})
stargazer.show_degrees_of_freedom(False)
stargazer.significant_digits(3)
stargazer.covariate_order(['z', 'hi', 'z_hi'])
stargazer.show_adj_r2 = False
stargazer.show_f_statistic = False
stargazer.show_residual_std_err = False
stargazer.show_notes = False

# adding custom rows with mean, sd, and p-values
stargazer.add_line('Mean DV', mean_std_df.loc['mean'].tolist(),location='fb')
stargazer.add_line('SD DV', mean_std_df.loc['std'].tolist(),location='fb')
stargazer.add_line('P-value T + TH = 0', pvals.loc['z+z_hi'].
 ↪tolist(),location='fb')
stargazer.add_line('P-value Treat', pvals.loc['z'].tolist(),location='fb')
stargazer.add_line('P-value Treat Bootstrap', bootstrap_pvals.loc['z'].
 ↪tolist(),location='fb')
stargazer.add_line('P-value High', pvals.loc['hi'].tolist(),location='fb')
stargazer.add_line('P-value High Bootstrap', bootstrap_pvals.loc['hi'].
 ↪tolist(),location='fb')
stargazer.add_line('P-value Treat x High', pvals.loc['z_hi'].tolist())
stargazer.add_line('P-value Treat x High Bootstrap', bootstrap_pvals.
 ↪loc['z_hi'].tolist(),location='fb')

latex_table7 = stargazer.render_latex()

# edit the latex table to add row for telling if Y1 Y2 or Pooled after \\[-1.
 ↪8ex] & (1) & (2) & (3) & (4) & (5) & (6) & (7) & (8) & (9) \\
latex_table7 = latex_table7.replace("\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) &␣
 ↪(6) & (7) & (8) & (9) \\",
                                    "\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) & (6)␣
 ↪& (7) & (8) & (9) \n \\\ & Y1 & Y2 & Pooled & Y1 & Y2 & Pooled & Y1 & Y2 &␣
 ↪Pooled \\")
latex_table7 = latex_table7.replace("\\begin{table}[!htbp] \\centering", "")
latex_table7 = latex_table7.replace("\\end{table}", "")

with open('tables/table7.tex','w') as file:
    file.write(latex_table7)
```

3.12.4 Creating Appendix Figure

[ ]:
```
# plot distribution of bootstrapped coefficients
fig, axs = plt.subplots(3, 3, figsize=(15, 15))
for i, dv in enumerate(['inventory_trim', 'netrevenue_trim',␣
 ↪'logtotcons_trim']):
    for j, treat in enumerate(['treat12', 'treat13', 'treatMS1MS2']):
        for k, var in enumerate(['z', 'hi', 'z_hi']):
            coef = bootstrap_ests[(dv, treat)][:, k]
            mu = np.mean(coef)
            sigma = np.std(coef)
            x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
            axs[i, j].hist(coef, bins=50, alpha=0.5, label=var, density=True)
            axs[i, j].plot(x, stats.t.pdf(x, df=ms1ms2_pooled_tab7['subloc'].
 ↪nunique()-1, loc=mu, scale=sigma))
            if j == 2:
                axs[i, j].set_title(f'{dv} - Pooled')
            else:
                axs[i, j].set_title(f'{dv} - Y{j+1}')
            axs[i, j].set_xlabel('Coefficient')
            axs[i, j].set_ylabel('Frequency')
            axs[i, j].legend()

plt.tight_layout()
plt.savefig('figures/boot_dist_tab7.png')
```


inventory_trim - Y1
inventory_trim - Y2
inventory_trim - Pooled
netrevenue_trim - Y1
netrevenue_trim - Y2
netrevenue_trim - Pooled
logtotcons_trim - Y1
logtotcons_trim - Y2
logtotcons_trim - Pooled
z
hi
z_hi
Frequency
Coefficient
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
−1.0
−0.5
0.0
0.5
1.0
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
−3000
−2000
−1000
0
1000
2000
3000
0
2
4
6
8
10
12
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
−0.2
−0.1
0.0
0.1
0.2
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

3.13 Table 8

```
[ ]: tab8_dt = ms1ms2_pooled.loc[:, ['treatMS1MS2', 'hi', 'treatMS1MS2hi',␣
      ↪'interviewdate', 'netrevenue_trim', 'strata_group', 'groupnum', 'subloc',␣
      ↪'Y1round1', 'Y1round2', 'Y1round3', 'Y2round1', 'Y2round2', 'Y2round3']].
      ↪dropna()
     tab8_dt['net_revenue_3'] = tab8_dt['netrevenue_trim'] * 3
     model = smf.ols('net_revenue_3 ~ treatMS1MS2 + hi + treatMS1MS2hi +␣
      ↪interviewdate + Y1round1 + Y1round2 + Y1round3 + Y2round1 + Y2round2 +␣
      ↪Y2round3', data=tab8_dt)
     results_t8 = model.fit(cov_type='cluster', cov_kwds={'groups':␣
      ↪tab8_dt['subloc']})
     model_params_t8 = results_t8.params
```

```
[ ]: results_t8.params
```

```
[ ]: Intercept       -986290.918662
     treatMS1MS2        3304.165930
     hi                  494.807445
     treatMS1MS2hi     -2450.309880
     interviewdate        58.265516
     Y1round1        -141751.795543
     Y1round2        -146510.601629
     Y1round3        -154677.306936
     Y2round1        -170728.707960
     Y2round2        -181748.455581
     Y2round3        -190874.051013
     dtype: float64
```

```
[ ]: results['model_5'].params*3
```

```
[ ]: Intercept       -1.128043e+06
     z                3.304166e+03
     hi               4.948074e+02
     z_hi            -2.450310e+03
     interviewdate    5.826552e+01
     Y1round2        -4.758806e+03
     Y1round3        -1.292551e+04
     Y2round1        -2.897691e+04
     Y2round2        -3.999666e+04
     Y2round3        -4.912226e+04
     dtype: float64
```

```
[ ]: # As specified in appendix B
     # Total population (HH) in the study area
     A1 = 7105.0
     # % of population in low saturation areas
     A2 = 0.5
     # % of population member of OAF
```

```
A3 = 0.3
# % of OAF members enrolled in study in a low saturation areas and b high saturation areas
A4a = 0.4
A4b = 0.8
# % in each sublocation assigned to receive treatment
A5 = 0.58

# Annualized coefficients
beta1 = results['model_5'].params['z']*3
beta2 = results['model_5'].params['hi']*3
beta3 = results['model_5'].params['z_hi']*3

table_8 = {
    "1. Direct gains/HH (Ksh)": [beta1, beta1+beta3],
    "2. Indirect gains/HH (Ksh)": [0, beta2],
    "3. Ratio of indirect to direct gains": [0, beta2 / (beta1+beta3)],
    "4. Direct beneficiary population (HH)": [A1*A2*A3*A4a*A5, A1*(1-A2)*A3*A4b*A5],
    "5. Total local population (HH)": [A1*A2, A1*(1-A2)]
}

# convert to DataFrame and perform final calculations
table_8_df = pd.DataFrame(table_8, index=["Low Saturation", "High Saturation"]).T

table_8_df.loc['6. Total direct gains (Ksh)'] = table_8_df.loc['1. Direct gains/HH (Ksh)']*table_8_df.loc['4. Direct beneficiary population (HH)']
table_8_df.loc['7. Total indirect gains (Ksh)'] = table_8_df.loc['2. Indirect gains/HH (Ksh)']*table_8_df.loc['4. Direct beneficiary population (HH)']
table_8_df.loc['8. Total gains (direct + indirect; Ksh)'] = table_8_df.loc['6. Total direct gains (Ksh)'] + table_8_df.loc['7. Total indirect gains (Ksh)']
table_8_df.loc['9. Fraction of gains direct'] = table_8_df.loc['6. Total direct gains (Ksh)'] / table_8_df.loc['8. Total gains (direct + indirect; Ksh)']
table_8_df.loc['10. Fraction of gains indirect'] = table_8_df.loc['7. Total indirect gains (Ksh)'] / table_8_df.loc['8. Total gains (direct + indirect; Ksh)']

table_8_df = table_8_df.map(lambda x: np.round(x, 3))

latex_table8 = table_8_df.to_latex(index=True, float_format="%.3f")
latex_table8 = latex_table8.replace('\\toprule', '\\\[-1.8ex]\hline \n \hline \\\[-1.8ex]')
latex_table8 = latex_table8.replace('\\bottomrule', '\\\[-1.8ex]\hline \n \hline \\\[-1.8ex]')
```

```python
with open('tables/table8.tex','w') as file:
    file.write(latex_table8)
```

proj03.py

April 26, 2024

[ ]:

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from scipy import stats

def cgmwildboot(data, model, n_bootstraps, cluster, bootcluster, seed=1234):
        """
        This function performs wild bootstrap inference for clustered data as␣
 ↪proposed by Cameron, Gelbach, and Miller (2008).

        Args:
        data: pandas DataFrame
        model: statsmodels regression model
        n_bootstraps: int, number of bootstrap samples
        cluster: list, name of the cluster variable to use in bootstrap␣
 ↪regressions
        bootcluster: list, name of the cluster variable to use in bootstrap␣
 ↪sampling
        seed: int, random seed

        Returns:
        b_ests: numpy array, bootstrapped parameter estimates
        b_pvals: numpy array, bootstrapped p-values
        """

        np.random.seed(seed)
        df = data.copy(deep=True)

        # gather dependent variable and independent variables from model.model.
 ↪formula
        dep = model.model.endog_names
        indep = model.model.exog_names[1:]
        b_ests = []
        b_pvals = []
        b_bse = []
        df['yhat'] = model.predict(df[indep])
        df['ehat'] = model.resid
        for i in range(n_bootstraps):
                # generate rademacher weights for each cluster
                signs = df[bootcluster].drop_duplicates().apply(lambda x: np.random.choice([-1, 1]))
                signs.index = df[bootcluster].drop_duplicates()
                df['sign'] = df[bootcluster].map(signs)
                # apply weights to residuals and add to predicted values
                df['we'] = df['ehat'] * df['sign']
                df['wy'] = df['yhat'] + df['we']
                df[dep] = df['wy']

                boot_model = smf.ols(model.model.formula, data=df).fit(cov_type='cluster', cov_kwds={'groups': df[cluster]})
                b_ests.append(boot_model.params)
                b_bse.append(boot_model.bse)

        # remove constant
        length = len(indep) + 1
        b_ests = np.array(b_ests)[:,1:length]
        b_bse = np.array(b_bse)[:,1:length]

        for i, var in enumerate(indep):

                # calculate the wald statistic for each variable
                w_boot = (b_ests[:,i]-model.params[var]) / b_bse[:,i]

                # here for simplicity we assume H0: beta = 0 as we do this for all variables, but should be adjusted if we want to generalize the function
                w = (model.params[var]-0) / model.bse[var]

                # calculate the p-value for the wald statistic
                pval = np.mean(np.abs(w_boot) > np.abs(w))
                b_pvals.append(pval)

        return b_ests, b_pvals
```