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)
print(ms1ms2_pooled_tab1.shape[0]) # checking we have the right number of␣
↪observations as described in the original article
```

```
1019
```

[ ]:
```
# 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_83627/2284489521.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)",
```

```python
    "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]')
print(latex_table1)
```

```
\begin{tabular}{lrrrrr}
\\[-1.8ex]\hline
 \hline \\[-1.8ex]
Baseline characteristic & Treat & Control & Obs & Std diff & P-val \\
\midrule
Male & 0.296 & 0.334 & 1589 & -0.081 & 0.107 \\
Number of adults & 3.004 & 3.196 & 1510 & -0.092 & 0.060 \\
```

```
Children in school & 2.998 & 3.072 & 1589 & -0.039 & 0.457 \\
Finished primary school & 0.718 & 0.772 & 1490 & -0.128 & 0.021 \\
Finished secondary school & 0.253 & 0.270 & 1490 & -0.039 & 0.458 \\
Total cropland (acres) & 2.441 & 2.398 & 1512 & 0.013 & 0.792 \\
Number of rooms in household & 3.073 & 3.252 & 1511 & -0.055 & 0.170 \\
Total school fees & 27239.693 & 29813.631 & 1589 & -0.064 & 0.182 \\
Average monthly consumption (Ksh) & 14970.862 & 15371.378 & 1437 & -0.033 &
0.554 \\
Average monthly consumption/capita (log) & 7.975 & 7.963 & 1434 & 0.019 & 0.721
\\
Total cash savings (Ksh) & 5157.396 & 8021.499 & 1572 & -0.095 & 0.013 \\
Total cash savings (trim) & 4731.623 & 5389.836 & 1572 & -0.046 & 0.327 \\
Has bank savings acct & 0.419 & 0.425 & 1589 & -0.012 & 0.815 \\
Taken bank loan & 0.079 & 0.083 & 1589 & -0.018 & 0.728 \\
Taken informal loan & 0.244 & 0.249 & 1589 & -0.011 & 0.836 \\
Liquid wealth (Ksh) & 93878.938 & 97280.922 & 1491 & -0.032 & 0.545 \\
Off-farm wages (Ksh) & 3916.817 & 3797.480 & 1589 & 0.009 & 0.849 \\
Business profit (Ksh) & 2302.588 & 1801.685 & 1589 & 0.083 & 0.322 \\
Avg $\%\Delta$ price Sep-Jun & 133.495 & 133.178 & 1504 & 0.004 & 0.939 \\
Expect $\%\Delta$ price Sep12-Jun13 & 124.680 & 117.255 & 1510 & 0.141 & 0.155
\\
2011 LR harvest (bags) & 9.364 & 9.025 & 1511 & 0.023 & 0.671 \\
Net revenue 2011 (Ksh) & -3303.691 & -4088.622 & 1428 & 0.031 & 0.749 \\
Net seller 2011 & 0.324 & 0.303 & 1428 & 0.047 & 0.394 \\
Autarkic 2011 & 0.068 & 0.060 & 1589 & 0.035 & 0.511 \\
\% maize lost 2011 & 0.016 & 0.013 & 1428 & 0.033 & 0.573 \\
2012 LR harvest (bags) & 11.181 & 11.030 & 1484 & 0.019 & 0.740 \\
Calculated interest correctly & 0.715 & 0.730 & 1580 & -0.035 & 0.504 \\
Digit span recall & 4.568 & 4.576 & 1504 & -0.007 & 0.891 \\
Maize giver & 0.261 & 0.261 & 1589 & -0.001 & 0.985 \\
\\[-1.8ex]\hline
 \hline \\[-1.8ex]
\end{tabular}
```

3.2 Running the model for tables 2 through 4

[ ]:
```python
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_83627/2854517969.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_83627/1060375524.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_83627/1060375524.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 Outputting the tables

Creating stagazar for table 2-4

[ ]:
```
latex_tables = []
for i, dv in enumerate(['inventory_trim', 'netrevenue_trim',␣
 ↪'logtotcons_trim']):
```

```python
    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'])

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

    latex_table = stargazer.render_latex()

    # general formatting
    latex_table = latex_table.replace("Adjusted $R^2$", "% Adjusted $R^2$")
    latex_table = latex_table.replace("Residual Std. Error", "% Residual Std. Error")
    latex_table = latex_table.replace("F Statistic", "% F Statistic")
    latex_table = latex_table.replace("\\textit{Note","% \\textit{Note")
    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 '
```

3.5.1 Table 2

```
[ ]: print(latex_tables[0])
```

```
\begin{tabular}{@{\extracolsep{5pt}}lcccccc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
& \multicolumn{6}{c}{\textit{Dependent variable: Inventory Trim}} \
\cr \cline{2-7}
\\[-1.8ex] & \multicolumn{2}{c}{Y1} & \multicolumn{2}{c}{Y2} &
\multicolumn{2}{c}{Pooled}  \\
\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) & (6)
 \\ & Overall & By rd & Overall & By rd & Overall & By rd \\
\hline \\[-1.8ex]
 Treat & 0.574$^{***}$ & & 0.546$^{***}$ & & 0.565$^{***}$ & \\
& (0.140) & & (0.129) & & (0.097) & \\
 Treat - R1 & & 0.872$^{***}$ & & 1.242$^{***}$ & & 1.050$^{***}$ \\
& & (0.276) & & (0.235) & & (0.184) \\
 Treat - R2 & & 0.753$^{***}$ & & 0.304$^{*}$ & & 0.546$^{***}$ \\
& & (0.171) & & (0.166) & & (0.120) \\
 Treat - R3 & & 0.111$^{}$ & & 0.082$^{}$ & & 0.094$^{}$ \\
& & (0.083) & & (0.344) & & (0.162) \\
 Mean DV & 3.021 & 3.021 & 1.959 & 1.959 & 2.56 & 2.56 \\
 SD DV & 3.726 & 3.726 & 3.089 & 3.089 & 3.503 & 3.503 \\
 P-Val Treat & <0.001 &  & <0.001 &  & <0.001 &  \\
 P-Val Treat FWER & <0.001 &  & <0.001 &  & <0.001 &  \\
 P-Val Treat - R1 &  & 0.002 &  & <0.001 &  & <0.001 \\
 P-Val Treat - R1 FWER &  & 0.004 &  & <0.001 &  & <0.001 \\
 P-Val Treat - R2 &  & <0.001 &  & 0.068 &  & <0.001 \\
 P-Val Treat - R2 FWER &  & <0.001 &  & 0.173 &  & <0.001 \\
 P-Val Treat - R3 &  & 0.183 &  & 0.812 &  & 0.561 \\
 P-Val Treat - R3 FWER &  & 0.33 &  & 0.913 &  & 0.631 \\
\hline \\[-1.8ex]
 Observations & 3836 & 3836 & 2944 & 2944 & 6780 & 6780 \\
 $R^2$ & 0.365 & 0.368 & 0.098 & 0.215 & 0.144 & 0.329 \\
 % Adjusted $R^2$ & 0.360 & 0.362 & 0.088 & 0.205 & 0.136 & 0.322 \\
 % Residual Std. Error & 2.982 (df=3803) & 2.975 (df=3799) & 2.950 (df=2911) &
2.754 (df=2907) & 3.256 (df=6716) & 2.884 (df=6710) \\
 % F Statistic & 69.431$^{***}$ (df=32; 3803) & 84.886$^{***}$ (df=36; 3799) &
64.996$^{***}$ (df=32; 2911) & 57.574$^{***}$ (df=36; 2907) & 52.508$^{***}$
```

```
(df=63; 6716) & 56.147$^{***}$ (df=69; 6710) \\
\hline
\hline \\[-1.8ex]
% \textit{Note:} & \multicolumn{6}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05;
$^{***}$p$<$0.01} \\
\end{tabular}
```

3.5.2 Table 3

[ ]:
```
print(latex_tables[1])
```

```
\begin{tabular}{@{\extracolsep{5pt}}lcccccc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
& \multicolumn{6}{c}{\textit{Dependent variable: Net Revenue Trim}} \
\cr \cline{2-7}
\\[-1.8ex] & \multicolumn{2}{c}{Y1} & \multicolumn{2}{c}{Y2} &
\multicolumn{2}{c}{Pooled}  \\
\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) & (6)
 \\ & Overall & By rd & Overall & By rd & Overall & By rd \\
\hline \\[-1.8ex]
 Treat & 263.790$^{}$ & & 854.114$^{***}$ & & 531.358$^{***}$ & \\
& (255.661) & & (303.802) & & (196.315) & \\
 Treat - R1 & & -1164.574$^{***}$ & & 16.478$^{}$ & & -613.581$^{**}$ \\
& & (322.956) & & (444.957) & & (271.653) \\
 Treat - R2 & & 509.851$^{}$ & & 1994.923$^{***}$ & & 1187.967$^{***}$ \\
& & (446.928) & & (503.696) & & (337.460) \\
 Treat - R3 & & 1370.344$^{***}$ & & 565.438$^{}$ & & 998.665$^{***}$ \\
& & (412.602) & & (403.307) & & (291.103) \\
 Mean DV & 485.812 & 485.812 & -2997.862 & -2997.862 & -1033.442 & -1033.442 \\
 SD DV & 6212.781 & 6212.781 & 6545.626 & 6545.626 & 6590.086 & 6590.086 \\
 P-Val Treat & 0.303 &  & 0.006 &  & 0.007 &  \\
 P-Val Treat FWER & 0.379 &  & 0.01 &  & 0.012 &  \\
 P-Val Treat - R1 &  & <0.001 &  & 0.971 &  & 0.024 \\
 P-Val Treat - R1 FWER &  & 0.002 &  & 0.971 &  & 0.044 \\
 P-Val Treat - R2 &  & 0.255 &  & <0.001 &  & <0.001 \\
 P-Val Treat - R2 FWER &  & 0.383 &  & <0.001 &  & 0.001 \\
 P-Val Treat - R3 &  & 0.001 &  & 0.163 &  & 0.001 \\
 P-Val Treat - R3 FWER &  & 0.003 &  & 0.259 &  & 0.001 \\
\hline \\[-1.8ex]
 Observations & 3795 & 3795 & 2935 & 2935 & 6730 & 6730 \\
 $R^2$ & 0.025 & 0.038 & 0.074 & 0.079 & 0.107 & 0.119 \\
 % Adjusted $R^2$ & 0.017 & 0.029 & 0.064 & 0.067 & 0.099 & 0.110 \\
 % Residual Std. Error & 6160.285 (df=3762) & 6123.381 (df=3758) & 6332.926
(df=2902) & 6321.436 (df=2898) & 6257.097 (df=6666) & 6218.002 (df=6660) \\
 % F Statistic & 4.652$^{***}$ (df=32; 3762) & 5.786$^{***}$ (df=36; 3758) &
```

```
30.217$^{***}$ (df=32; 2902) & 29.094$^{***}$ (df=36; 2898) & 27.156$^{***}$
(df=63; 6666) & 10478661843.432$^{***}$ (df=69; 6660) \\
\hline
\hline \\[-1.8ex]
% \textit{Note:} & \multicolumn{6}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05;
$^{***}$p$<$0.01} \\
\end{tabular}
```

3.5.3 Table 4

```
[ ]: print(latex_tables[2])
```

```
\begin{tabular}{@{\extracolsep{5pt}}lcccccc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
& \multicolumn{6}{c}{\textit{Dependent variable: Log Total HH Consumption Trim}}
\
\cr \cline{2-7}
\\[-1.8ex] & \multicolumn{2}{c}{Y1} & \multicolumn{2}{c}{Y2} &
\multicolumn{2}{c}{Pooled}  \\
\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) & (6)
 \\ & Overall & By rd & Overall & By rd & Overall & By rd \\
\hline \\[-1.8ex]
 Treat & 0.012$^{}$ & & 0.064$^{*}$ & & 0.036$^{}$ & \\
& (0.030) & & (0.036) & & (0.023) & \\
 Treat - R1 & & -0.033$^{}$ & & 0.064$^{}$ & & 0.013$^{}$ \\
& & (0.047) & & (0.047) & & (0.033) \\
 Treat - R2 & & 0.028$^{}$ & & 0.076$^{*}$ & & 0.049$^{*}$ \\
& & (0.039) & & (0.043) & & (0.029) \\
 Treat - R3 & & 0.038$^{}$ & & 0.052$^{}$ & & 0.044$^{}$ \\
& & (0.042) & & (0.047) & & (0.031) \\
 Mean DV & 9.477 & 9.477 & 9.653 & 9.653 & 9.554 & 9.554 \\
 SD DV & 0.621 & 0.621 & 0.652 & 0.652 & 0.64 & 0.64 \\
 P-Val Treat & 0.683 &  & 0.082 &  & 0.127 &  \\
 P-Val Treat FWER & 0.683 &  & 0.103 &  & 0.127 &  \\
 P-Val Treat - R1 &  & 0.487 &  & 0.173 &  & 0.687 \\
 P-Val Treat - R1 FWER &  & 0.487 &  & 0.259 &  & 0.687 \\
 P-Val Treat - R2 &  & 0.481 &  & 0.077 &  & 0.089 \\
 P-Val Treat - R2 FWER &  & 0.487 &  & 0.173 &  & 0.133 \\
 P-Val Treat - R3 &  & 0.365 &  & 0.272 &  & 0.164 \\
 P-Val Treat - R3 FWER &  & 0.469 &  & 0.349 &  & 0.211 \\
\hline \\[-1.8ex]
 Observations & 3792 & 3792 & 2944 & 2944 & 6736 & 6736 \\
 $R^2$ & 0.026 & 0.027 & 0.051 & 0.053 & 0.055 & 0.056 \\
 % Adjusted $R^2$ & 0.018 & 0.018 & 0.041 & 0.041 & 0.046 & 0.046 \\
 % Residual Std. Error & 0.615 (df=3759) & 0.615 (df=3755) & 0.638 (df=2911) &
```

```
0.638 (df=2907) & 0.625 (df=6672) & 0.625 (df=6666) \\
 % F Statistic & 21.960$^{***}$ (df=32; 3759) & 21.858$^{***}$ (df=36; 3755) &
250.735$^{***}$ (df=32; 2911) & 65.033$^{***}$ (df=36; 2907) & 23.128$^{***}$
(df=63; 6672) & 21.830$^{***}$ (df=69; 6666) \\
\hline
\hline \\[-1.8ex]
% \textit{Note:} & \multicolumn{6}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05;
$^{***}$p$<$0.01} \\
\end{tabular}
```

3.5.4 Table 5

[ ]:
```
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'])

# adding p-values
stargazer.add_line('Mean DV', mean_df.
 ↪loc[['netsales_trim','netsales_trim','purchase','sales'],'treatMS1MS2'].
 ↪tolist())
stargazer.add_line('SD DV', std_df.
 ↪loc[['netsales_trim','netsales_trim','purchase','sales'],'treatMS1MS2'].
 ↪tolist())
eff_p_val = ['','']+pval.loc[['purchase','sales'],'treatMS1MS2'].tolist()
eff_p_val_fwer = ['','']+pval_fwer.loc[['purchase','sales'],'treatMS1MS2_fwer'].
 ↪tolist()

stargazer.add_line('P-Val Treat', ['','']+pval.
 ↪loc[['purchase','sales'],'treatMS1MS2'].tolist())
stargazer.add_line('P-Val Treat FWER', ['','']+pval_fwer.
 ↪loc[['purchase','sales'],'treatMS1MS2_fwer'].tolist())

latex_table5 = stargazer.render_latex()

# general formatting
latex_table5 = latex_table5.replace("Adjusted $R^2$", "% Adjusted $R^2$")
latex_table5 = latex_table5.replace("Residual Std. Error", "% Residual Std.␣
 ↪Error")
```

```python
latex_table5 = latex_table5.replace("F Statistic", "% F Statistic")
latex_table5 = latex_table5.replace("\\textit{Note","% \\textit{Note")
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 \\")

print(latex_table5)
```

```
\begin{tabular}{@{\extracolsep{5pt}}lcccc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
\\[-1.8ex] & \multicolumn{2}{c}{Net Sales} & \multicolumn{2}{c}{Effective Price}
\\
\\[-1.8ex] & (1) & (2) & (3) & (4)
 \\ & Overall & By rd & Purchase & Sales \\
\hline \\[-1.8ex]
 Treat & 0.186$^{***}$ & & -57.449$^{**}$ & 145.509$^{***}$ \\
& (0.063) & & (27.156) & (41.767) \\
 Treat - R1 & & -0.205$^{**}$ & & \\
& & (0.096) & & \\
 Treat - R2 & & 0.384$^{***}$ & & \\
& & (0.104) & & \\
 Treat - R3 & & 0.369$^{***}$ & & \\
& & (0.092) & & \\
 Mean DV & -0.421 & -0.421 & 3085.372 & 2809.763 \\
 SD DV & 2.038 & 2.038 & 534.511 & 504.822 \\
 P-Val Treat &  &  & 0.035 & 0.001 \\
 P-Val Treat FWER &  &  & 0.044 & 0.001 \\
\hline \\[-1.8ex]
 Observations & 6740 & 6740 & 2014 & 1428 \\
 $R^2$ & 0.098 & 0.102 & 0.089 & 0.066 \\
 % Adjusted $R^2$ & 0.089 & 0.093 & 0.060 & 0.024 \\
 % Residual Std. Error & 2.020 (df=6672) & 2.016 (df=6670) & 639.432 (df=1951) &
789.099 (df=1365) \\
 % F Statistic & 20.619$^{***}$ (df=67; 6672) & 370.686$^{***}$ (df=69; 6670) &
34.249$^{***}$ (df=62; 1951) & 13.002$^{***}$ (df=62; 1365) \\
\hline
\hline \\[-1.8ex]
% \textit{Note:} & \multicolumn{4}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05;
$^{***}$p$<$0.01} \\
\end{tabular}
```

```
/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'])

# adding custom rows with p-values
stargazer.add_line('P-value High', pvals.loc['hi'].values.tolist())
```

```python
stargazer.add_line('P-value High Bootstrap', bootstrap_pvals.loc['hi'].values.
  ↪tolist())
stargazer.add_line('P-value Month', pvals.loc['monthnum'].values.tolist())
stargazer.add_line('P-value High Bootstrap', bootstrap_pvals.loc['monthnum'].
  ↪values.tolist())
stargazer.add_line('P-value High x Month', pvals.loc['interact'].values.
  ↪tolist())
stargazer.add_line('P-value High x Month Bootstrap', bootstrap_pvals.
  ↪loc['interact'].values.tolist())

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("Adjusted $R^2$", "% Adjusted $R^2$")
latex_table6 = latex_table6.replace("Residual Std. Error", "% Residual Std.␣
  ↪Error")
latex_table6 = latex_table6.replace("F Statistic", "% F Statistic")
latex_table6 = latex_table6.replace("\\textit{Note","% \\textit{Note")
latex_table6 = latex_table6.replace("salesPrice_trim_norm","Trimmed Sales␣
  ↪Price")
latex_table6 = latex_table6.replace("\\begin{table}[!htbp] \\centering", "")
latex_table6 = latex_table6.replace("\\end{table}", "")

print(latex_table6)
```

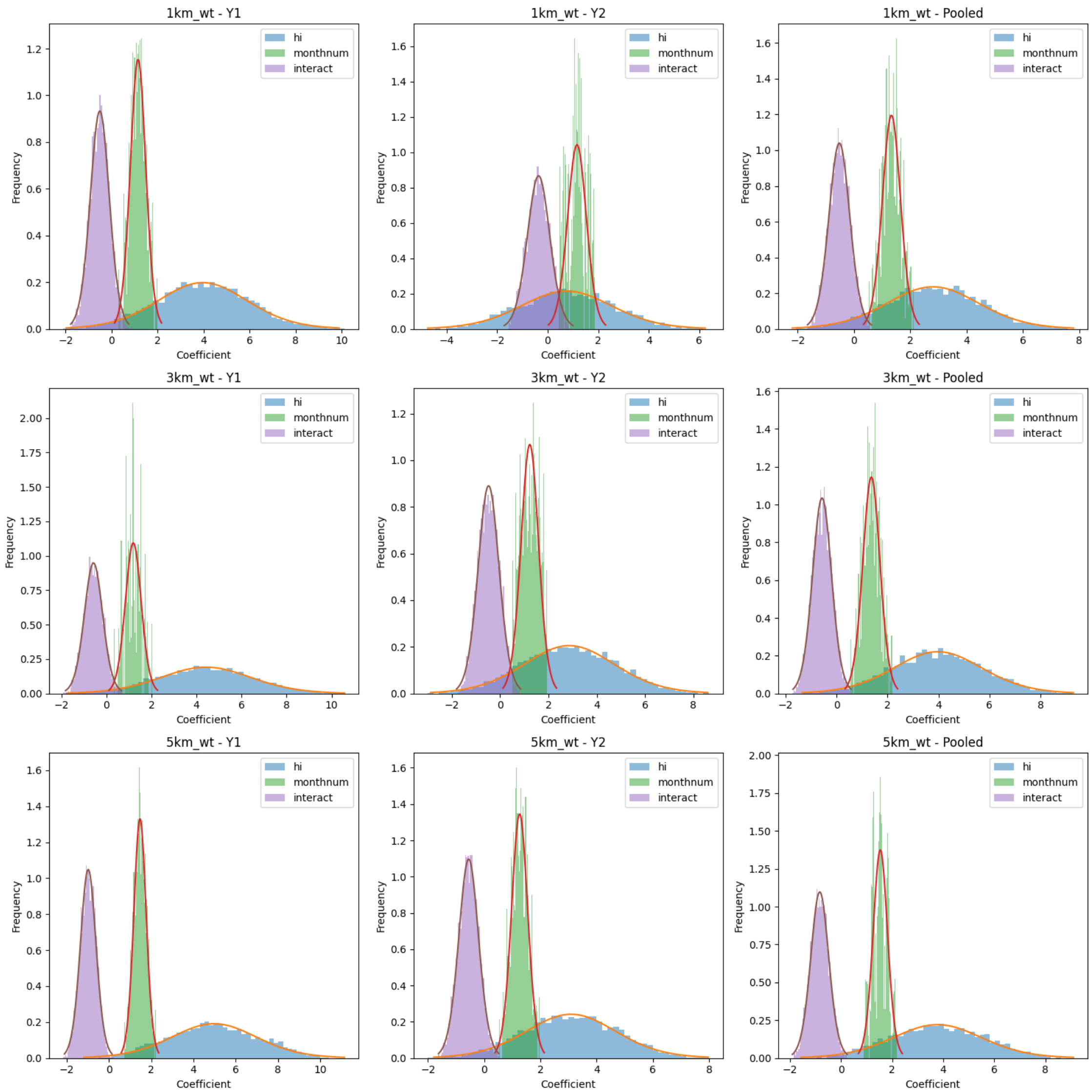
```
\begin{tabular}{@{\extracolsep{5pt}}lccccc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
& \multicolumn{5}{c}{\textit{Dependent variable: Trimmed Sales Price}} \
\cr \cline{2-6}
\\[-1.8ex] & \multicolumn{3}{c}{Main Specification (3km)} &
\multicolumn{2}{c}{Robustness (Pooled)}  \\
\\[-1.8ex] & (1) & (2) & (3) & (4) & (5)
 \\ & Y1 & Y2 & Pooled & 1km & 5km \\
\hline \\[-1.8ex]
 High & 4.410$^{**}$ & 2.855$^{}$ & 3.970$^{**}$ & 2.787$^{}$ & 3.766$^{**}$ \\
& (2.091) & (1.992) & (1.817) & (1.719) & (1.822) \\
 Month & 1.189$^{***}$ & 1.224$^{***}$ & 1.364$^{***}$ & 1.327$^{***}$ &
1.537$^{***}$ \\
& (0.363) & (0.377) & (0.350) & (0.339) & (0.291) \\
 High x Month & -0.574$^{}$ & -0.476$^{}$ & -0.573$^{}$ & -0.520$^{}$ &
-0.835$^{**}$ \\
```

```
& (0.422) & (0.459) & (0.386) & (0.390) & (0.366) \\
 P-value High & 0.051 & 0.171 & 0.044 & 0.124 & 0.056 \\
 P-value High Bootstrap & 0.082 & 0.197 & 0.083 & 0.152 & 0.096 \\
 P-value Month & 0.005 & 0.005 & 0.001 & 0.001 & <0.001 \\
 P-value High Bootstrap & 0.033 & <0.001 & 0.026 & 0.016 & <0.001 \\
 P-value High x Month & 0.192 & 0.315 & 0.158 & 0.2 & 0.038 \\
 P-value High x Month Bootstrap & 0.223 & 0.345 & 0.192 & 0.227 & 0.061 \\
\hline \\[-1.8ex]
 Observations & 491 & 381 & 872 & 872 & 872 \\
 $R^2$ & 0.077 & 0.031 & 0.058 & 0.055 & 0.060 \\
 % Adjusted $R^2$ & 0.071 & 0.023 & 0.055 & 0.052 & 0.056 \\
 % Residual Std. Error & 10.071 & 14.651 & 12.700 & 12.726 & 12.685 \\
 % F Statistic & 6.401$^{***}$ & 7.496$^{***}$ & 13.411$^{***}$ & 10.971$^{***}$
& 16.730$^{***}$ \\
\hline
\hline \\[-1.8ex]
% \textit{Note:} & \multicolumn{5}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05;
$^{***}$p$<$0.01} \\
\end{tabular}
```

3.11.1 Create Appendix figure

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

latex_table7 = stargazer.render_latex()
```

```python
# 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("Adjusted $R^2$", "% Adjusted $R^2$")
latex_table7 = latex_table7.replace("Residual Std. Error", "% Residual Std.␣
↪Error")
latex_table7 = latex_table7.replace("F Statistic", "% F Statistic")
latex_table7 = latex_table7.replace("\\textit{","% \\textit{")
latex_table7 = latex_table7.replace("\\begin{table}[!htbp] \\centering", "")
latex_table7 = latex_table7.replace("\\end{table}", "")

print(latex_table7)
```

```
\begin{tabular}{@{\extracolsep{5pt}}lccccccccc}
\\[-1.8ex]\hline
\hline \\[-1.8ex]
\\[-1.8ex] & \multicolumn{3}{c}{Inventory} & \multicolumn{3}{c}{Net Revenues} &
\multicolumn{3}{c}{Consumption}  \\
\\[-1.8ex] & (1) & (2) & (3) & (4) & (5) & (6) & (7) & (8) & (9)
 \\ & Y1 & Y2 & Pooled & Y1 & Y2 & Pooled & Y1 & Y2 & Pooled \\
\hline \\[-1.8ex]
 Treat & 0.759$^{***}$ & 0.546$^{***}$ & 0.740$^{***}$ & 1059.602$^{**}$ &
1193.768$^{*}$ & 1101.389$^{**}$ & 0.012$^{}$ & -0.051$^{}$ & -0.011$^{}$ \\
& (0.189) & (0.185) & (0.155) & (437.732) & (685.048) & (430.091) & (0.040) &
(0.040) & (0.023) \\
 High & 0.124$^{}$ & -0.028$^{}$ & 0.017$^{}$ & 533.903$^{}$ & -152.603$^{}$ &
164.936$^{}$ & -0.003$^{}$ & -0.084$^{}$ & -0.047$^{}$ \\
& (0.355) & (0.219) & (0.241) & (551.179) & (558.948) & (479.685) & (0.051) &
(0.053) & (0.043) \\
 Treat x High & -0.333$^{}$ & -0.065$^{}$ & -0.291$^{}$ & -1114.628$^{**}$ &
-555.215$^{}$ & -816.770$^{}$ & -0.013$^{}$ & 0.174$^{***}$ & 0.067$^{*}$ \\
& (0.229) & (0.255) & (0.192) & (535.594) & (804.864) & (520.036) & (0.052) &
(0.055) & (0.037) \\
 Mean DV & 2.646 & 1.68 & 2.143 & 310.264 & -3434.378 & -1650.216 & 9.476 &
9.614 & 9.548 \\
 SD DV & 3.52 & 2.871 & 3.234 & 6087.188 & 6093.296 & 6370.068 & 0.633 & 0.631 &
0.636 \\
 P-value T + TH = 0 & 0.006 & 0.015 & 0.006 & 0.864 & 0.146 & 0.408 & 0.97 &
0.006 & 0.081 \\
 P-value Treat & 0.001 & 0.01 & <0.001 & 0.028 & 0.102 & 0.021 & 0.767 & 0.228 &
0.627 \\
 P-value Treat Bootstrap & <0.001 & <0.001 & <0.001 & 0.061 & 0.129 & 0.059 &
```

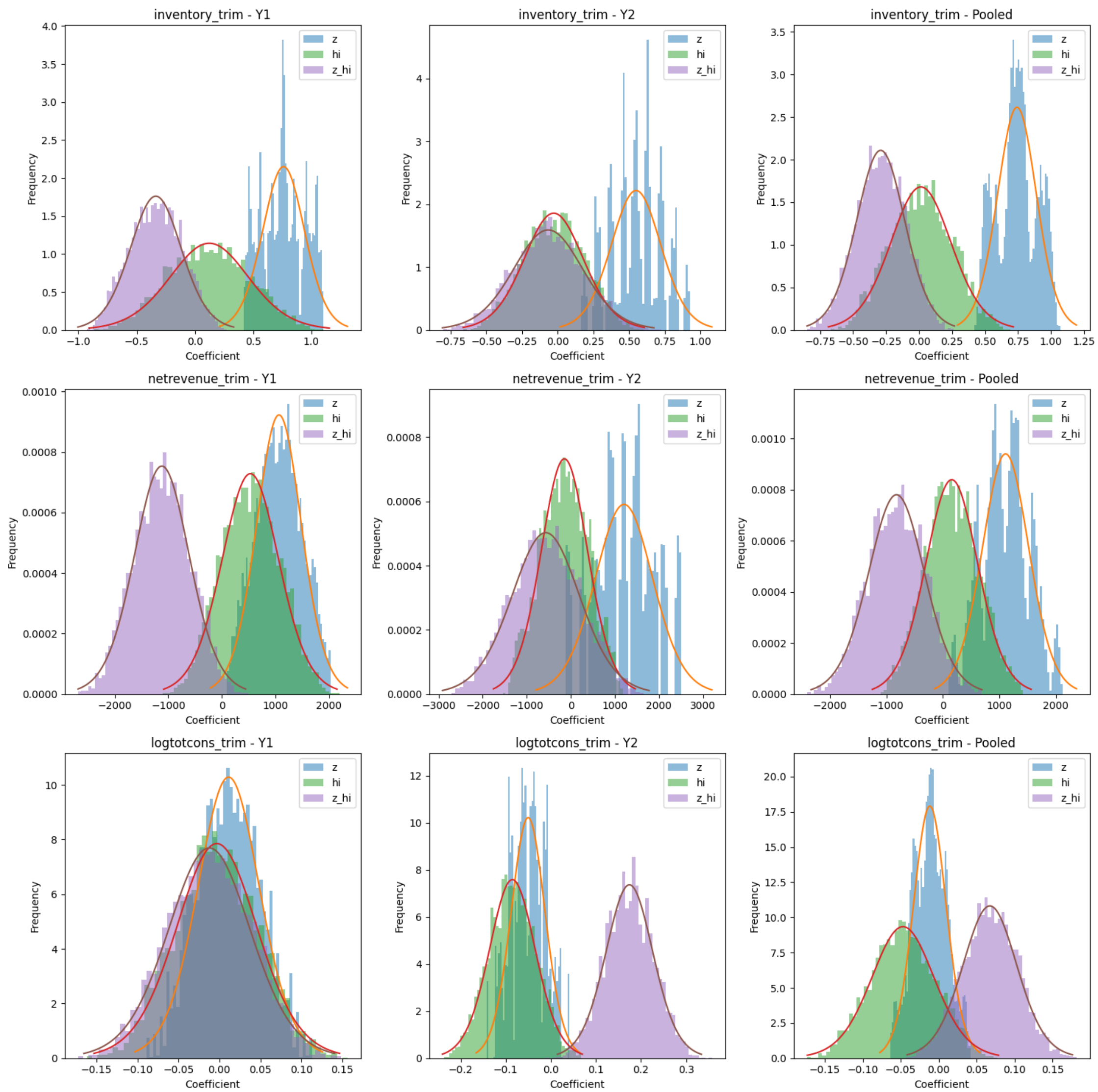
```
0.758 & 0.212 & 0.614 \\
 P-value High & 0.731 & 0.901 & 0.945 & 0.347 & 0.789 & 0.735 & 0.962 & 0.136 &
0.295 \\
 P-value High Bootstrap & 0.764 & 0.911 & 0.951 & 0.39 & 0.804 & 0.766 & 0.962 &
0.139 & 0.313 \\
 P-value Treat x High & 0.165 & 0.802 & 0.149 & 0.054 & 0.501 & 0.136 & 0.802 &
0.007 & 0.091 \\
 P-value Treat x High Bootstrap & 0.213 & 0.791 & 0.166 & 0.073 & 0.51 & 0.144 &
0.803 & 0.005 & 0.095 \\
\hline \\[-1.8ex]
 Observations & 3836 & 2944 & 6780 & 3795 & 2935 & 6730 & 3792 & 2944 & 6736 \\
 $R^2$ & 0.346 & 0.184 & 0.293 & 0.009 & 0.043 & 0.091 & 0.002 & 0.017 & 0.025
\\
 % Adjusted $R^2$ & 0.345 & 0.182 & 0.292 & 0.008 & 0.041 & 0.090 & 0.000 &
0.015 & 0.024 \\
 % Residual Std. Error & 3.015 & 2.793 & 2.947 & 6188.647 & 6410.741 & 6286.767
& 0.621 & 0.647 & 0.633 \\
 % F Statistic & 369.556$^{***}$ & 93.029$^{***}$ & 364.779$^{***}$ &
2.004$^{*}$ & 19.627$^{***}$ & 119.335$^{***}$ & 0.616$^{}$ & 4.496$^{***}$ &
16.477$^{***}$ \\
\hline
\hline \\[-1.8ex]
% \textit{Note:} & \multicolumn{9}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05;
$^{***}$p$<$0.01} \\
\end{tabular}
```

3.12.4 Creating Appendix Figure

[ ]:
```python
# 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')
```



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
```

[ ]:
```
# Annualized coefficients
treat_coef = model_params_t8['treatMS1MS2']
treat_hi_coef = (model_params_t8['treatMS1MS2'] +␣
↪model_params_t8['treatMS1MS2hi'])
hi_coef = model_params_t8['hi']

# Direct beneficiary population
direct_beneficiary_pop_low = 247.0
direct_beneficiary_pop_high = 495.0

# Total direct gains
total_direct_gains_low = treat_coef * direct_beneficiary_pop_low
total_direct_gains_high = treat_hi_coef * direct_beneficiary_pop_high

# Total indirect gains (only applicable to high saturation areas)
total_indirect_gains_high = hi_coef * 3553.0

# Total gains
total_gains_low = total_direct_gains_low
total_gains_high = total_direct_gains_high + total_indirect_gains_high

# Fraction of gains direct
fraction_gains_direct_low = 1  # All gains are direct in low saturation
fraction_gains_direct_high = total_direct_gains_high / total_gains_high

# Fraction of gains indirect (only applicable to high saturation areas)
fraction_gains_indirect_high = total_indirect_gains_high / total_gains_high

table_8 = {
    "1. Direct gains/HH (Ksh)": [treat_coef, treat_hi_coef],
    "2. Indirect gains/HH (Ksh)": [0, hi_coef],
    "3. Ratio of indirect to direct gains": [0, hi_coef / treat_hi_coef],
    "4. Direct beneficiary population (HH)": [direct_beneficiary_pop_low,␣
↪direct_beneficiary_pop_high],
    "5. Total local population (HH)": [3553.0, 3553.0],
    "6. Total direct gains (Ksh)": [total_direct_gains_low,␣
↪total_direct_gains_high],
    "7. Total indirect gains (Ksh)": [0, total_indirect_gains_high],
    "8. Total gains (direct + indirect; Ksh)": [total_gains_low,␣
↪total_gains_high],
```

```python
    "9. Fraction of gains direct": [fraction_gains_direct_low,␣
↪fraction_gains_direct_high],
    "10. Fraction of gains indirect": [0, fraction_gains_indirect_high],
}

# Convert the calculations to DataFrame and transpose it
table_8_df = pd.DataFrame(table_8, index=["Low Saturation", "High Saturation"]).
↪T

# Now you can print table_8_df to see the recreated table
table_8_df

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]')
print(latex_table8)
```

```
\begin{tabular}{lrr}
\\[-1.8ex]\hline
 \hline \\[-1.8ex]
 & Low Saturation & High Saturation \\
\midrule
1. Direct gains/HH (Ksh) & 3304.166 & 853.856 \\
2. Indirect gains/HH (Ksh) & 0.000 & 494.807 \\
3. Ratio of indirect to direct gains & 0.000 & 0.579 \\
4. Direct beneficiary population (HH) & 247.000 & 495.000 \\
5. Total local population (HH) & 3553.000 & 3553.000 \\
6. Total direct gains (Ksh) & 816128.985 & 422658.745 \\
7. Total indirect gains (Ksh) & 0.000 & 1758050.851 \\
8. Total gains (direct + indirect; Ksh) & 816128.985 & 2180709.596 \\
9. Fraction of gains direct & 1.000 & 0.194 \\
10. Fraction of gains indirect & 0.000 & 0.806 \\
\\[-1.8ex]\hline
 \hline \\[-1.8ex]
\end{tabular}
```

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):
        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
```