```
import sys
#Open log
sys.stdout = open("PSet_1_ARE213/PSet1_Q5_ARE213.log", 'w')

# Import packages
import numpy as np
import pandas as pd
import statsmodels.api as sm

from sklearn.linear_model import LogisticRegression

#Set wd + import df
os.chdir("/Users/maxsnyder/Dropbox/Berkeley ARE/Year 2/First Semester/
Applied Metrics/Metrics PS1/ARE213_Fall2023")
df = pd.read_csv("PSet_1_ARE213/clean_pset1.csv")

################
#5.a
################

#Logit to estimate propensity score
y_log_reg = df['tobacco']

#cor with y and D
x1 = ['alcohol', 'mrace3_2', 'mrace3_3', 'ormothhis', 'adeq_2.0',
'adeq_3.0', 'cardiac', 'pre4000', 'phyper',
     'diabetes', 'anemia', 'lung', 'dlivord', 'educ_0.0', 'educ_1.0',
'educ_2.0','dmage', 'dmar','tot_2.0',

'tot_3.0','tot_4.0','tot_5.0','tot_6.0','tot_7.0','tot_8.0','live_1.0'
, 'live_2.0', 'live_3.0','live_4.0',
      'live_5.0','live_6.0','live_7.0','live_8.0', 'live_9.0']
#cor with D not y
x2 = []
#cor with y not D
x3 = ['dgestat', 'csex', 'plur_1']


X_log_reg = df[x1+x3]

model = LogisticRegression(solver='liblinear', random_state=0)
model.fit(X_log_reg, y_log_reg)

#2nd column gives us predictions
predictions = model.predict_proba(X_log_reg)[:,1]

#Calculate weights
wt = (df['tobacco'] / predictions) + (1 - df['tobacco']) /(1 -
predictions)
```

```
#Demeaned X matrix
for item in x1+x3:
    df[item+'demeaned'] = df[item].sub(df[item].mean())

#multiply by D
for item in x1+x3:
    df['tobacco*'+item] = df[item+'demeaned']*df['tobacco']


#concatenate
d_times_demeaned_X = df[['tobacco*alcohol',
       'tobacco*mrace3_2', 'tobacco*mrace3_3', 'tobacco*ormothhis',
       'tobacco*adeq_2.0', 'tobacco*adeq_3.0', 'tobacco*cardiac',
       'tobacco*pre4000', 'tobacco*phyper',
       'tobacco*diabetes', 'tobacco*anemia', 'tobacco*lung',
'tobacco*dlivord',
       'tobacco*educ_0.0', 'tobacco*educ_1.0', 'tobacco*educ_2.0',
       'tobacco*dgestat', 'tobacco*dmage', 'tobacco*dmar',
'tobacco*csex',
       'tobacco*tot_2.0', 'tobacco*tot_3.0', 'tobacco*tot_4.0',
       'tobacco*tot_5.0', 'tobacco*tot_6.0', 'tobacco*tot_7.0',
       'tobacco*tot_8.0', 'tobacco*live_1.0', 'tobacco*live_2.0',
       'tobacco*live_3.0', 'tobacco*live_4.0', 'tobacco*live_5.0',
       'tobacco*live_6.0', 'tobacco*live_7.0', 'tobacco*live_8.0',
       'tobacco*live_9.0', 'tobacco*plur_1']]


#Outcome: birthweight
y_log_reg = df['dbrwt']

#Create final covariates matrix:
double_robust_reg_X = pd.concat([df['tobacco'],
                                 X_log_reg,
                                 d_times_demeaned_X], axis=1)


fit_wls = sm.WLS(y_log_reg, double_robust_reg_X, weights=wt).fit()
print("5A Results")
print(fit_wls.summary())



################
#5.b
################

#https://www.kirenz.com/post/2019-08-12-python-lasso-regression-auto/
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold


poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_log_reg)

#How many covars at the start?
print("How many covars once we include interaction terms?")
print(np.shape(X_poly))


y_log_reg = df['dbrwt']

model = Lasso()
# define model evaluation method
cv = RepeatedKFold(n_splits=2, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['alpha'] = np.arange(0, 1, 0.001)
# This section takes a very long time to run! Commenting after alpha
estimation
# search = GridSearchCV(model, grid,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1, verbose=10)
# # perform the search
# results = search.fit(X_poly, y_log_reg)
# # summarize
# print('MAE: %.3f' % results.best_score_)
# print('Config: %s' % results.best_params_)

#Hard coding alpha value
reg = Lasso(alpha=.136, tol=1, normalize=True)
reg.fit(X_poly, y_log_reg)

#Drop coefs that are smaller then .001
keep_coef_after_lasso = pd.Series((reg.coef_ > .001))

X_poly_df = pd.DataFrame(X_poly)
x_a_tilde = (X_poly_df[X_poly_df.columns[keep_coef_after_lasso]])

#Repeat the same process for x_b_tilde

y_log_reg = df['tobacco']

# This section takes a very long time to run! Commenting out after
alpha estimation
# grid = dict()
# grid['alpha'] = np.arange(0, .001, 0.0001)
# # define search
# search = GridSearchCV(model, grid,
```

```
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1, verbose=10)
# # perform the search
# results = search.fit(X_poly, y_log_reg)


#Hard coding the alpha
reg = Lasso(alpha=0.0001, tol=1, normalize=True)
reg.fit(X_poly, y_log_reg)
keep_coef_after_lasso = pd.Series((reg.coef_ > .00001))

x_b_tilde = (X_poly_df[X_poly_df.columns[keep_coef_after_lasso]])

df['intercept'] = 1

#Create final covar matrix with D and union of x_a_tilde and x_b_tilde
final_covars = pd.concat([df['intercept'],  df['tobacco'], x_a_tilde,
x_b_tilde], axis=1)

import statsmodels.api as sm
y_log_reg = df['dbrwt']
ols = sm.OLS(y_log_reg, final_covars)
ols_result = ols.fit()
print("5B Results Long Covars")
print(ols_result.summary())


#Create final covar matrix with D and union of just x_a_tilde to test
sensitivity of inclusion of x_b_tilde
just_x_a_tilde_covars = pd.concat([df['intercept'],  df['tobacco'],
x_a_tilde], axis=1)
ols = sm.OLS(y_log_reg, just_x_a_tilde_covars)
ols_result = ols.fit()
print("5B Results Short Covars (just x_a_tilde)")
print(ols_result.summary())

#How many overlapped?

print("How many x_a_tilde elements?")
print(x_a_tilde.columns)

print("How many x_b_tilde elements?")
print(x_b_tilde.columns)

print("How many overlapping elements?")
print(list(np.intersect1d(x_a_tilde.columns, x_b_tilde.columns)))
#0 elements.

#Close log
sys.stdout.close()
print("hi")
```