survival_analysis

May 15, 2020

[1]:
```
# store start time to get execution time of entire script
import time
start_time = time.time()
```

[2]:
```
# helper functions for displaying table data

import numpy as np
from IPython.display import display_html

# n is the number of columns to display data in
def display_side_by_side(series_obj, n):
    df = pd.DataFrame(series_obj)
    partition = int(round(len(df) / n))
    lower_bound = 0
    upper_bound = partition
    args = []
    for i in range(n):
        args.append(df[lower_bound:upper_bound])
        lower_bound += partition
        upper_bound += partition
    helper(args)

def helper(args):
    html_str=''
    for df in args:
        html_str+=df.to_html()
    display_html(html_str.replace('table','table style="display:
 ↪inline"'),raw=True)
```

[3]:
```
# helper function for plotting out ground truth curves

import matplotlib.pyplot as plt
plt.rcParams["font.weight"] = "bold"

def get_ground_truth(data):
    relapsed = data[data.Illicit_Cens5 == 1]
    counts = relapsed['Illicit_Days5'].value_counts()
```

```
    counts = counts.to_dict()
    temp = [len(data)] * 365
    labels = list(range(365))
    for i in range(365):
        labels[i] += 1
    total = 0
    errors = []
    for i in range(365):
        try:
            temp[i] = temp[i] - counts[i+1] - total
            total = total + counts[i+1]
        except KeyError:
            errors.append(i)

    for ele in sorted(errors, reverse = False):
        if ele != 0:
            temp[ele] = temp[ele-1]
        else:
             temp[0] = len(data)
    temp = [x / len(data) for x in temp]
    return labels, temp
```

[4]:
```
from sklearn.model_selection import cross_validate
from sksurv.ensemble import GradientBoostingSurvivalAnalysis
from sksurv.ensemble import RandomSurvivalForest
from sksurv.linear_model import CoxnetSurvivalAnalysis

def run_models(X, y, label):
    rsf = RandomSurvivalForest()
    scores = cross_validate(rsf, X, y, cv=5)
    rsf_score = scores['test_score'].mean()
    print('RF score:', rsf_score)

    rsf = RandomSurvivalForest()
    rsf.fit(X, y)

    # l1_ratio = 1 adjusts model to implement LASSO method for penalties
    rcr = CoxnetSurvivalAnalysis(l1_ratio=1)
    # one-hot encode all variables (except primsev) to get hazards across␣
 ↪groups, drop highest reference group
    lasso_X = get_lasso_features(X)

    scores = cross_validate(rcr, lasso_X, y, cv=5)
    rcr_score = scores['test_score'].mean()
    print('Lasso score:', rcr_score)
```

```
    # fit_baseline_model = True allows us to create survival/hazard plots after␣
↪model is fit
    rcr = CoxnetSurvivalAnalysis(fit_baseline_model=True, l1_ratio=1)
    rcr.fit(lasso_X, y)

    # concordance index
    scores = {'Model': ['Random Forest','Lasso','Dataset Size'],
              label: [rsf_score,rcr_score,int(X.shape[0])]}

    concordance = pd.DataFrame(data=scores)

    # return scores and models
    return concordance, rsf, rcr
```

```
//anaconda3/lib/python3.7/importlib/_bootstrap.py:219: RuntimeWarning:
sklearn.tree._splitter.Splitter size changed, may indicate binary
incompatibility. Expected 360 from C header, got 368 from PyObject
  return f(*args, **kwds)
```

[5]:
```
def get_survival_graph(rsf, rcr, X, Y, label, filename):
    pred_surv_rsf = rsf.predict_survival_function(X)

    # one-hot encode all variables (except primsev) to get hazards across␣
↪groups, drop highest reference group
    lasso_X = get_lasso_features(X)

    pred_surv_rcr = rcr.predict_survival_function(lasso_X)

    # display survival plot
    plt.suptitle(label)
    plt.plot(np.mean([person for person in pred_surv_rsf], axis=0), label='RF')
    plt.plot(np.mean([person.y for person in pred_surv_rcr], axis=0),␣
↪label='Lasso')
    labels, temp = get_ground_truth(Y)
    plt.plot(labels, temp, label='Ground Truth')
    plt.legend()
    plt.xlim(0, 365)
    plt.xticks(np.arange(0, 365, step=50))
    plt.yticks(np.arange(0, 1.1, step=0.1))
    plt.savefig(filename)

    plt.show()
```

[6]:
```
from tqdm.notebook import tqdm # used to show progress bar

def get_feature_importance(X, y, rsf, rcr, label):
    # feature importances from Random Forest
```

```
    feature_importance_rf = pd.DataFrame({'Feature':list(X.columns),})
    feature_importance_rf[label] = 0

    scores = cross_validate(rsf, X, y, cv=5)
    reference = scores['test_score'].mean()

    for i,row in tqdm(feature_importance_rf.iterrows(),␣
↪total=feature_importance_rf.shape[0]):
        feat = row['Feature']
        temp_data = X.copy()
        temp_data[feat] = np.random.permutation(temp_data[feat].values)
        temp_scores = cross_validate(rsf, temp_data, y, cv=5)
        temp_score = temp_scores['test_score'].mean()
        percent_change = (reference - temp_score) / reference * 100 # percent␣
↪change
        if percent_change < 0:
            percent_change = 0 # removing feature helped model, should not be␣
↪reflected in feature importance
        feature_importance_rf.iloc[i, feature_importance_rf.columns.
↪get_loc(label)] = percent_change

    feature_importance_rf = feature_importance_rf.nlargest(10,[label]) # keep␣
↪top 10 features

    # feature importances from Lasso
    lasso_X = get_lasso_features(X)
    feature_importance_lasso = pd.DataFrame({'Feature':list(lasso_X.columns),
                                             label:np.average(rcr.coef_,␣
↪weights=rcr.alphas_, axis = 1),})
    # remove features that were zero-ed out by lasso
    feature_importance_lasso =␣
↪feature_importance_lasso[feature_importance_lasso[label] != 0]
    # convert regression coefficients to hazard ratios
    feature_importance_lasso[label] = np.exp(feature_importance_lasso[label])
    # rank by magnitude of deviation from 1
    feature_importance_lasso[label + '_adjusted'] = np.
↪absolute(feature_importance_lasso[label]-1)
    feature_importance_lasso = feature_importance_lasso.nlargest(10,[label +␣
↪'_adjusted']) # keep top 10 features

    return feature_importance_rf, feature_importance_lasso
```

[7]:
```
def get_lasso_features(X):
    features_to_ignore =␣
↪['female_cd','nonwhite_cd','unemplmt_cd','SUDSy','primsev_cd_1','primsev_cd_2',
```

```
                                ␣
↪'primsev_cd_3','primsev_cd_4','primsev_cd_5','primsev_cd_6']
    lasso_X = X.copy()
    for col in lasso_X.columns:
        if col not in features_to_ignore:
            one_hot = pd.get_dummies(lasso_X[col], prefix=col)
            one_hot = one_hot.loc[:, ~one_hot.columns.str.endswith('0')] # drop␣
↪group and use as reference
            lasso_X = lasso_X.drop(col,axis = 1)
            lasso_X = lasso_X.join(one_hot)
    #print(lasso_X.columns)
    return lasso_X
```

Survival Analysis by Severity

[8]:
```
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
import csv

df = pd.read_csv('data/data_superset.csv')
df.head()
```

[8]:

	Unnamed: 0	Unnamed: 0.1	Unnamed: 0.1.1	ID	State	City	\
0	0	1	2	929	OH	Cleveland	
1	1	2	3	951	OH	Cleveland	
2	2	3	4	1032	OH	Cleveland	
3	3	18	19	1673	KY	Louisville	
4	4	21	22	3870	AZ	Tucson	

	agyaddr	xobsyr_0	Illicit_Days5	Illicit_Cens5	\
0	1276 West Third St. #400	2006	354	0	
1	1276 West Third St. #400	2006	365	0	
2	1276 West Third St. #400	2006	365	0	
3	1220 Bardstown Rd	2006	365	0	
4	3130 E Broadway Blvd	2006	5	1	

	female_cd	nonwhite_cd	unemplmt_cd	prsatx_cd	gvsg_cd	CWSg_0_cd	\
0	0	0	0	0	1	0	
1	0	0	0	0	0	0	
2	0	0	0	0	2	0	
3	0	0	0	0	0	0	
4	0	0	0	1	2	1	

	srprobg_cd	dssg_0_cd	epsg_0_cd	adhdg_0_cd	cdsg_0_cd	cjsig_0_cd	\
0	1	0	1	0	1	1	
1	1	0	0	0	0	1	

```
2           1          1          1          1          1          0
3           0          0          1          1          0          0
4           2          2          2          1          2          1

   lrig_0_cd  srig_0_cd  SESg_0_cd  r4ag_0_cd  primsev_cd_1  primsev_cd_2  \
0          0          1          0          2             1             0
1          0          1          0          2             0             0
2          2          1          0          2             0             0
3          2          2          0          0             0             0
4          1          2          2          2             0             0

   primsev_cd_3  primsev_cd_4  primsev_cd_5  primsev_cd_6  B2a_0g  SUDSy_0  \
0             0             0             0             0       0        3
1             1             0             0             0       0        2
2             1             0             0             0       0        2
3             1             0             0             0       0        2
4             1             0             0             0       0       11

                                  Address        lat         lng state_name  \
0  1276 West Third St. #400, Cleveland, OH  41.501028  -81.697772       Ohio
1  1276 West Third St. #400, Cleveland, OH  41.501028  -81.697772       Ohio
2  1276 West Third St. #400, Cleveland, OH  41.501028  -81.697772       Ohio
3       1220 Bardstown Rd, Louisville, KY  38.236398  -85.717815   Kentucky
4        3130 E Broadway Blvd, Tucson, AZ  32.221465 -110.926070    Arizona

   county_FIPS    block_FIPS  murder_numg  %_dropoutg  %_povertyg  \
0      39035.0  3.903511e+14            0         0.0         0.0
1      39035.0  3.903511e+14            0         0.0         0.0
2      39035.0  3.903511e+14            0         0.0         0.0
3      21111.0  2.111101e+14            0         0.0         0.0
4       4019.0  4.019002e+13            0         0.0         0.0

   %_public_assistanceg  %_unemployedg closest  gran  \
0                   0.0            0.0     NaN   0.0
1                   0.0            0.0     NaN   0.0
2                   0.0            0.0     NaN   0.0
3                   0.0            0.0     NaN   0.0
4                   0.0            0.0     NaN   1.0

                                  point  pop_deng
0   ('41.501028000000005', '-81.697772')       0.0
1   ('41.501028000000005', '-81.697772')       0.0
2   ('41.501028000000005', '-81.697772')       0.0
3  ('38.236397499999995', '-85.7178152')       0.0
4  ('32.2214651', '-110.92607029999999')       0.0
```

[9]:
```
df['r4ag_0_cd'].value_counts()
```

[9]:
```
2    4225
0    4108
1    1735
Name: r4ag_0_cd, dtype: int64
```

[10]:
```
# drop unnecessary columns
cols_to_drop = ['Address','lat','lng','xobsyr_0','Unnamed: 0','Unnamed: 0.
 ↪1','Unnamed: 0.1.1',
                ␣
 ↪'ID','State','City','agyaddr','state_name','gran','srprobg_cd','county_FIPS','block_FIPS',
                 'point','closest']

df.drop(columns=cols_to_drop, inplace=True)
df.dropna(inplace=True) # drops any remaining rows with null values

# uncomment to get CONTROL statistics
#cols_to_drop =␣
 ↪['pop_deng','%_dropoutg','%_unemployedg','%_public_assistanceg','%_povertyg','murder_numg']
#df.drop(columns=cols_to_drop, inplace=True)

df = df.astype(int)
df = df.sample(frac=1).reset_index(drop=True) # shuffle rows
df.shape
```

[10]:
```
(10068, 31)
```

[11]:
```
df.head()
```

[11]:

	Illicit_Days5	Illicit_Cens5	female_cd	nonwhite_cd	unemplmt_cd	\
0	364	0	0	1	0	
1	98	0	0	0	0	
2	188	0	0	1	0	
3	30	1	0	1	0	
4	356	0	1	0	0	

	prsatx_cd	gvsg_cd	CWSg_0_cd	dssg_0_cd	epsg_0_cd	adhdg_0_cd	cdsg_0_cd	\
0	0	0	0	2	1	2	1	
1	0	0	0	2	1	2	1	
2	0	0	1	1	1	1	1	
3	0	1	0	1	1	1	0	
4	0	2	0	2	2	2	2	

	cjsig_0_cd	lrig_0_cd	srig_0_cd	SESg_0_cd	r4ag_0_cd	primsev_cd_1	\
0	0	2	2	0	2	0	
1	1	1	2	0	1	1	
2	2	1	2	1	2	1	
3	0	1	1	0	2	0	

```
4           0          1          2          0          2              0

   primsev_cd_2  primsev_cd_3  primsev_cd_4  primsev_cd_5  primsev_cd_6  \
0             0             1             0             0             0
1             0             0             0             0             0
2             0             0             0             0             0
3             0             1             0             0             0
4             0             1             0             0             0

   B2a_0g  SUDSy_0  murder_numg  %_dropoutg  %_povertyg  %_public_assistanceg  \
0       2        8            0           0           0                     0
1       1       10            2           0           0                     0
2       1        3            0           0           0                     0
3       0        0            0           0           0                     0
4       0        2            0           0           0                     0

   %_unemployedg  pop_deng
0              0         0
1              0         2
2              0         0
3              0         0
4              0         0
```

Full Population Survival Analysis

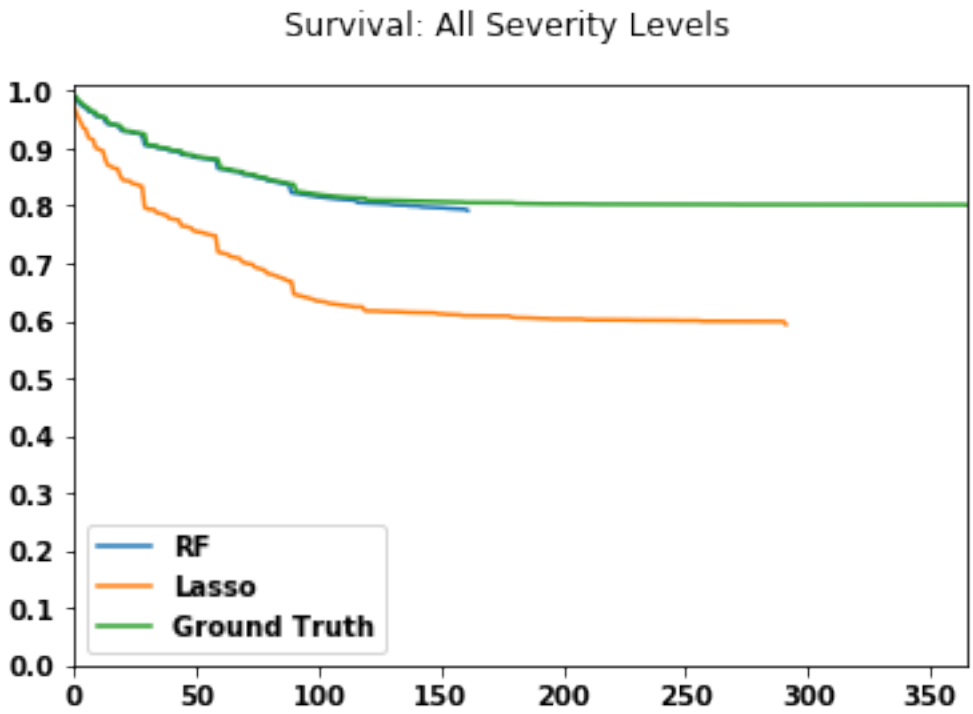
[12]:
```
from sklearn.model_selection import train_test_split
from sksurv.util import Surv

predictor_var = 'Illicit_Days5'
censoring_var = 'Illicit_Cens5'

X = df.copy()
Y = X[[censoring_var, predictor_var]]
X.drop(columns=[censoring_var, predictor_var], inplace=True)
y = Surv.from_arrays(Y[censoring_var], Y[predictor_var]) # structured array to␣
 ↪ensure correct censoring

print(X.shape, y.shape)
```

```
(10068, 29) (10068,)
```

[13]:
```
%%time
full_concordance, rsf, rcr = run_models(X, y, 'ALL')
```

```
RF score: 0.6744597173047271
Lasso score: 0.6818463753475946
CPU times: user 26.7 s, sys: 2.39 s, total: 29.1 s
Wall time: 28.8 s
```

[14]:
```
get_survival_graph(rsf, rcr, X, Y, 'Survival: All Severity Levels','graphs/
↪survival_all.png')
```



Subclinical Severity Survival Analysis

[15]:
```
X = df[df.SUDSy_0 < 2]
Y = X[[censoring_var, predictor_var]]
X.drop(columns=[censoring_var, predictor_var, 'SUDSy_0'], inplace=True)

y = Surv.from_arrays(Y[censoring_var], Y[predictor_var]) # structured array to␣
↪ensure correct censoring

print(X.shape, y.shape)
```

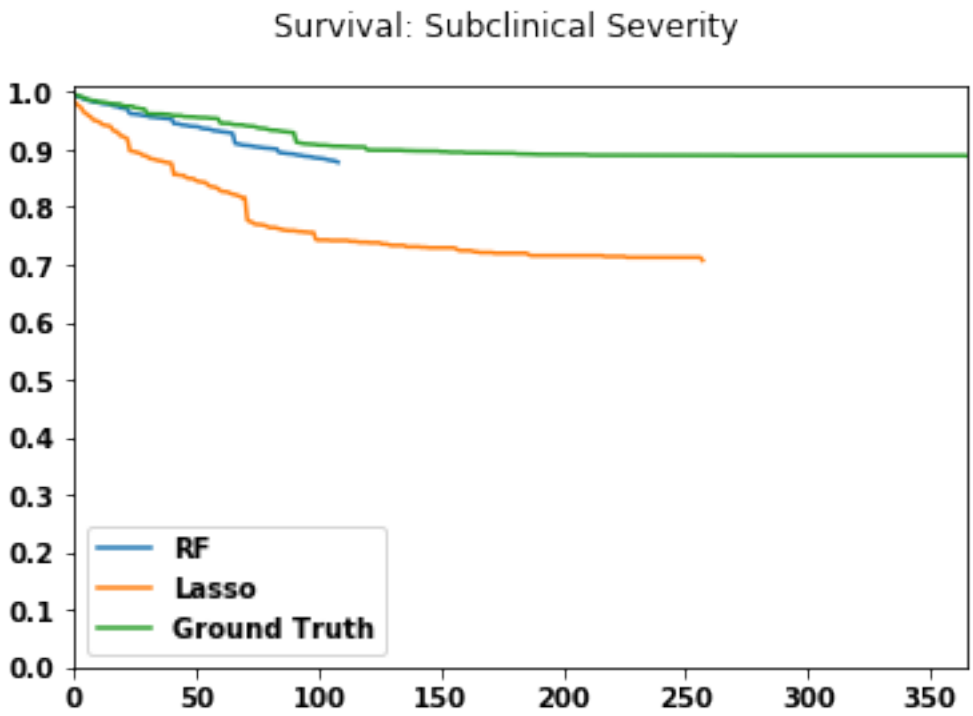
```
(3250, 28) (3250,)
```

```
//anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4097:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

[16]:
```
%%time
subclinical_concordance, rsf, rcr = run_models(X, y, 'SUB')
```

```
RF score: 0.654662869761251
Lasso score: 0.6850586806437148
CPU times: user 8.94 s, sys: 860 ms, total: 9.8 s
Wall time: 11.1 s
```

[17]:
```
get_survival_graph(rsf, rcr, X, Y, 'Survival: Subclinical Severity', 'graphs/
↪survival_subclinical.png')
```



[18]:
```
%%time
subclinical_feature_importance_rf, subclinical_feature_importance_lasso = \
                                        get_feature_importance(X, y, rsf, rcr,␣
↪'SUB')
```

```
HBox(children=(IntProgress(value=0, max=28), HTML(value='')))


CPU times: user 2min 41s, sys: 7.11 s, total: 2min 48s
Wall time: 2min 53s
```

Mild/Moderate Severity Survival Analysis

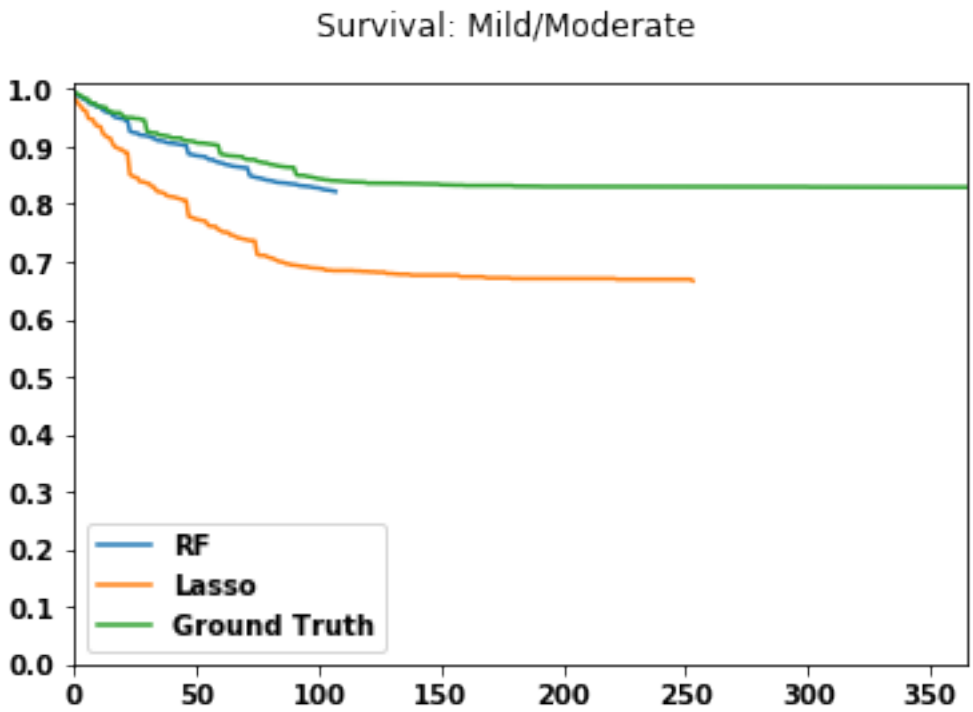
[19]:
```
X = df[df.SUDSy_0 >= 2]
X = X[X.SUDSy_0 <= 5]
Y = X[[censoring_var, predictor_var]]
X.drop(columns=[censoring_var, predictor_var, 'SUDSy_0'], inplace=True)

y = Surv.from_arrays(Y[censoring_var], Y[predictor_var]) # structured array to␣
↪ensure correct censoring

print(X.shape, y.shape)
```

```
(2838, 28) (2838,)
```

[20]:
```
%%time
mild_concordance, rsf, rcr = run_models(X, y, 'MILD')
```

```
RF score: 0.5750388120263283
Lasso score: 0.5986931261055546
CPU times: user 7.01 s, sys: 425 ms, total: 7.43 s
Wall time: 6.8 s
```

[21]:
```
get_survival_graph(rsf, rcr, X, Y, 'Survival: Mild/Moderate', 'graphs/
↪survival_mild.png')
```



[22]:
```
%%time
mild_feature_importance_rf, mild_feature_importance_lasso =␣
↪get_feature_importance(X, y, rsf, rcr, 'MILD')
```

```
HBox(children=(IntProgress(value=0, max=28), HTML(value='')))


CPU times: user 2min 26s, sys: 6.86 s, total: 2min 33s
Wall time: 2min 36s
```

Severe Severity Survival Analysis

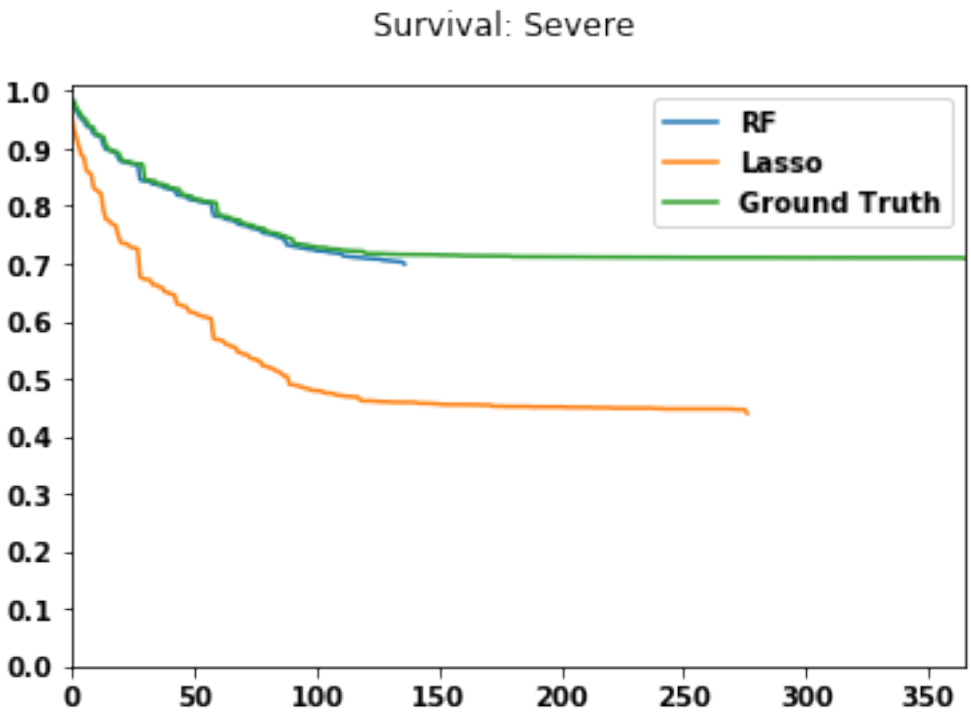
[23]:
```
X = df[df.SUDSy_0 > 5]
Y = X[[censoring_var, predictor_var]]
X.drop(columns=[censoring_var, predictor_var, 'SUDSy_0'], inplace=True)

y = Surv.from_arrays(Y[censoring_var], Y[predictor_var]) # structured array to␣
↪ensure correct censoring

print(X.shape, y.shape)
```

```
(3980, 28) (3980,)

//anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4097:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

[24]:
```
%%time
severe_concordance, rsf, rcr = run_models(X, y, 'SEVERE')
```

```
RF score: 0.6103601856123729
Lasso score: 0.6165492278881434
CPU times: user 9.6 s, sys: 816 ms, total: 10.4 s
Wall time: 9.62 s
```

[25]:
```
get_survival_graph(rsf, rcr, X, Y, 'Survival: Severe', 'graphs/survival_severe.
↪png')
```



[26]:
```
%%time
severe_feature_importance_rf, severe_feature_importance_lasso =␣
↪get_feature_importance(X, y, rsf, rcr, 'SEVERE')
```

```
HBox(children=(IntProgress(value=0, max=28), HTML(value='')))


CPU times: user 3min 27s, sys: 17.8 s, total: 3min 45s
Wall time: 3min 51s
```

Overall Statistics

[27]:
```
overall_concordance = pd.concat([subclinical_concordance,␣
↪mild_concordance['MILD'], severe_concordance['SEVERE'],
                                  full_concordance['ALL']], axis=1)
pd.DataFrame(data=overall_concordance).round(4)
```

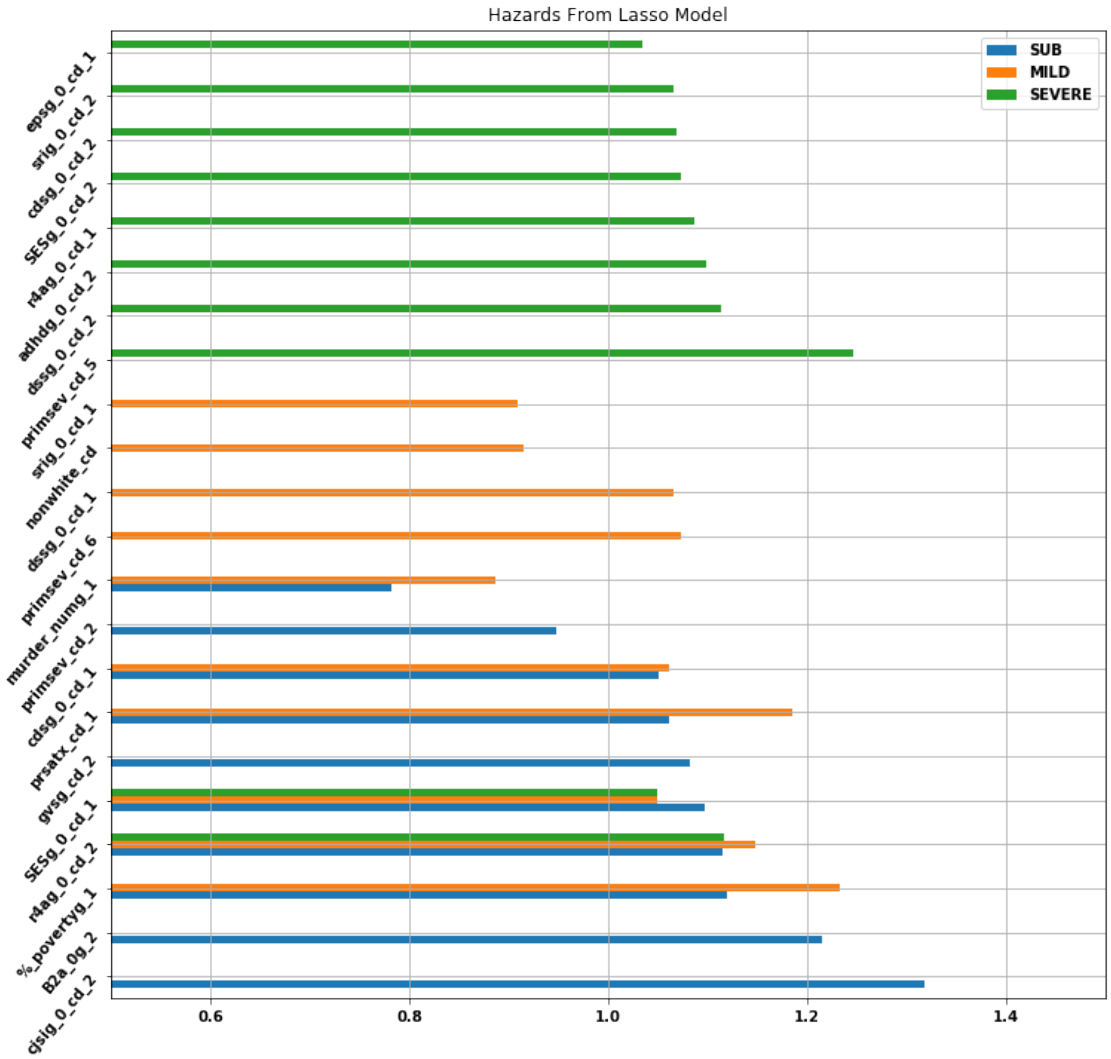
[27]:

	Model	SUB	MILD	SEVERE	ALL
0	Random Forest	0.6547	0.5750	0.6104	0.6745
1	Lasso	0.6851	0.5987	0.6165	0.6818
2	Dataset Size	3250.0000	2838.0000	3980.0000	10068.0000

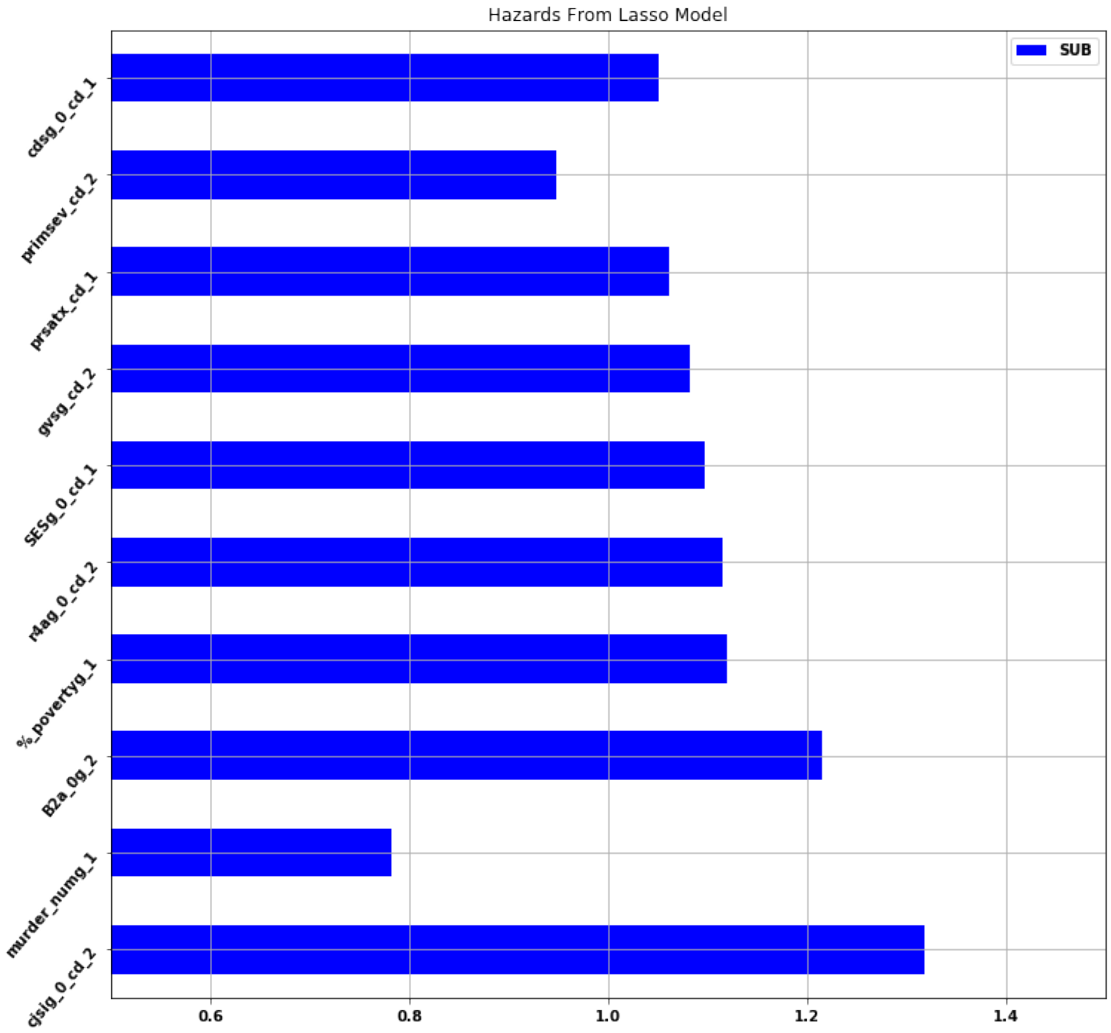
[28]:
```
overall_feature_importance_lasso = pd.
↪merge(subclinical_feature_importance_lasso, \
```

```
                                                mild_feature_importance_lasso,␣
 ↪on='Feature', how='outer')
overall_feature_importance_lasso = pd.merge(overall_feature_importance_lasso, \
                                                severe_feature_importance_lasso,␣
 ↪on='Feature', how='outer')
overall_feature_importance_lasso.fillna(0, inplace=True)
display_side_by_side(overall_feature_importance_lasso, 2)
```

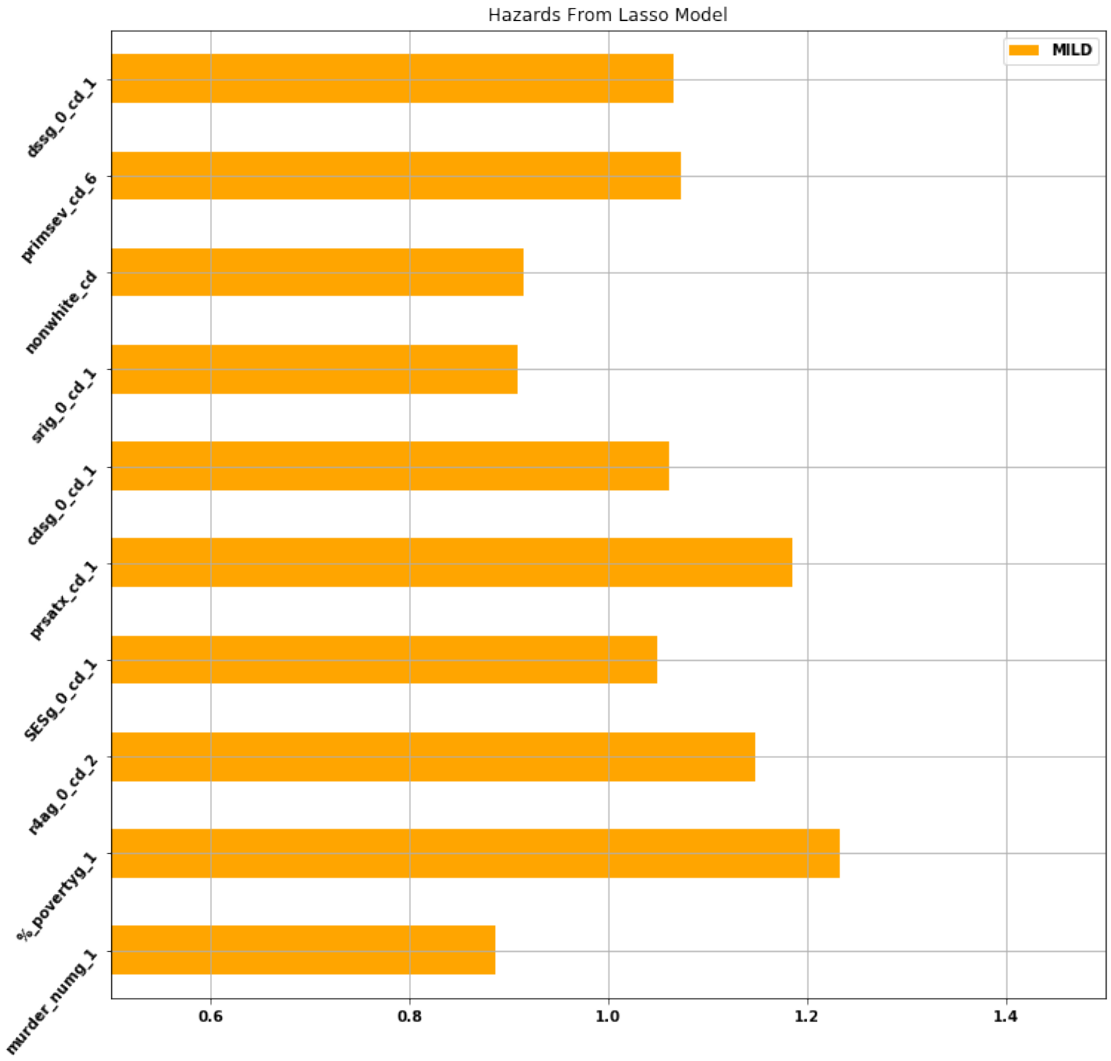
[29]:
```
haz_df = pd.DataFrame({'SUB': overall_feature_importance_lasso['SUB'].tolist(),
                   'MILD': overall_feature_importance_lasso['MILD'].tolist(),
                   'SEVERE': overall_feature_importance_lasso['SEVERE'].
 ↪tolist()},
                  index=overall_feature_importance_lasso['Feature'].tolist())
haz_df = haz_df.replace(1, 0)
haz_df.sort_values(by=['SUB','MILD','SEVERE'], ascending=False, inplace=True)
ax = haz_df.plot.barh(rot=50, figsize=(12, 12))
ax.set_xlim([0.5,1.5])
ax.grid()
ax.set_title('Hazards From Lasso Model')
fig = ax.get_figure()

fig.savefig('graphs/hazards_lasso.png', bbox_inches='tight')
```



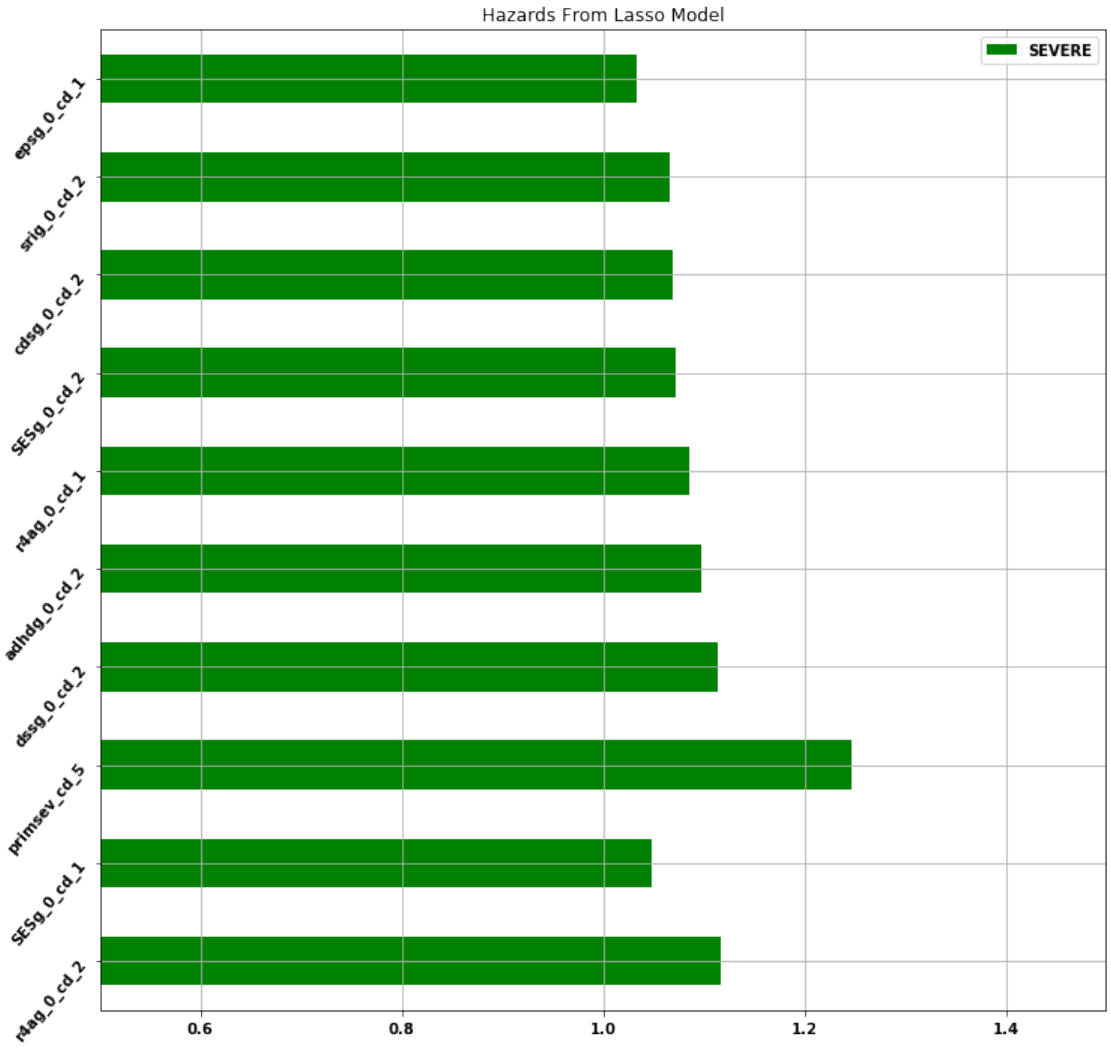
[30]:
```
haz_sub = pd.DataFrame({'SUB': overall_feature_importance_lasso['SUB'].
 ↪tolist()},
                  index=overall_feature_importance_lasso['Feature'].tolist())
haz_sub = haz_sub[haz_sub.SUB != 0]
haz_df.sort_values(by=['SUB'], ascending=False, inplace=True)

ax = haz_sub.plot.barh(rot=50, figsize=(12, 12), color='blue')
ax.set_xlim([0.5,1.5])
ax.grid()
ax.set_title('Hazards From Lasso Model')
fig = ax.get_figure()

fig.savefig('graphs/hazards_lasso_sub.png', bbox_inches='tight')
```



[31]:
```
haz_mild = pd.DataFrame({'MILD': overall_feature_importance_lasso['MILD'].
 ↪tolist()},
                  index=overall_feature_importance_lasso['Feature'].tolist())
haz_mild = haz_mild[haz_mild.MILD != 0]
haz_df.sort_values(by=['MILD'], ascending=False, inplace=True)

ax = haz_mild.plot.barh(rot=50, figsize=(12, 12), color='orange')
ax.set_xlim([0.5,1.5])
ax.grid()
ax.set_title('Hazards From Lasso Model')
fig = ax.get_figure()

fig.savefig('graphs/hazards_lasso_mild.png', bbox_inches='tight')
```



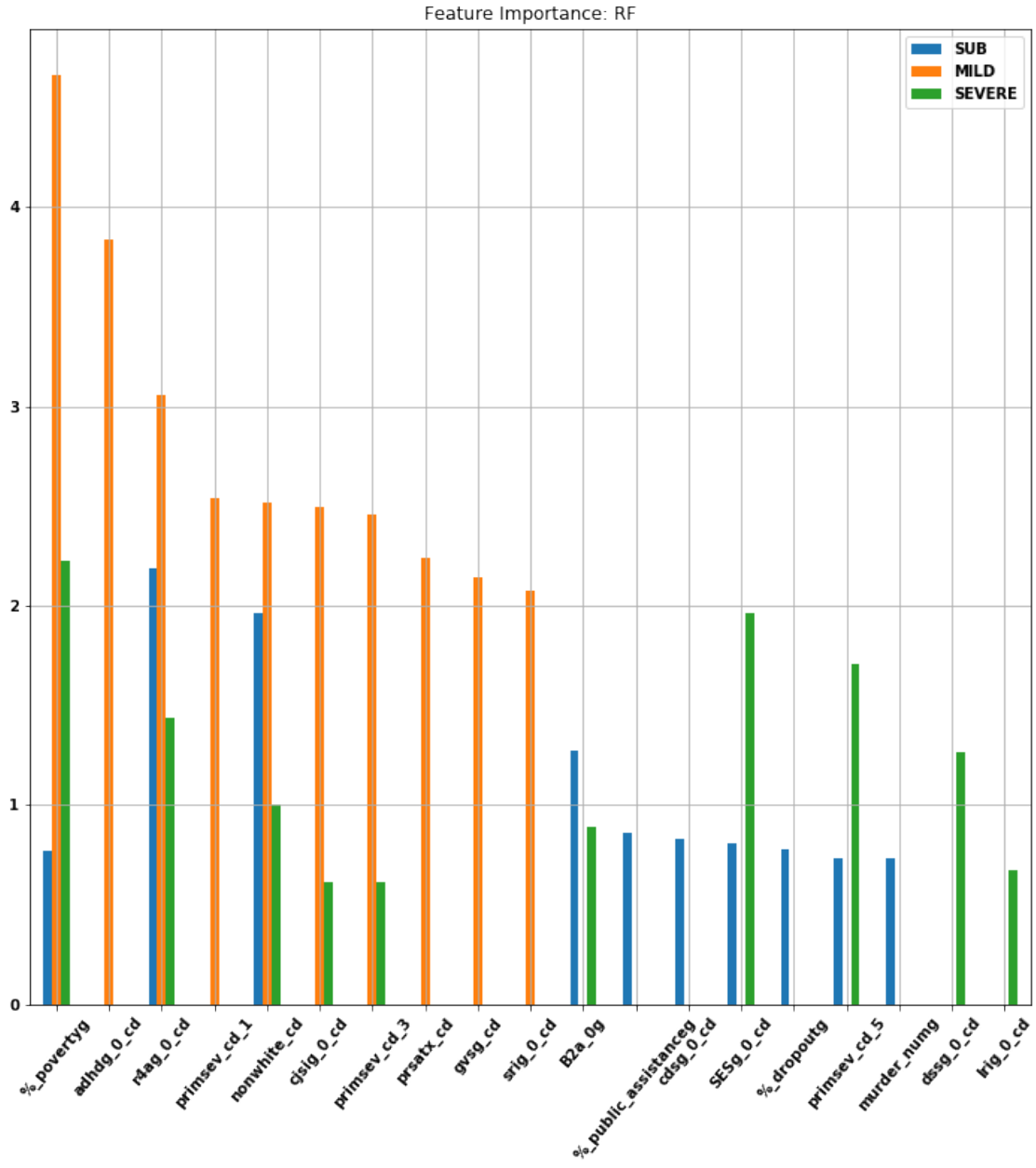
[32]:
```
haz_severe = pd.DataFrame({'SEVERE': overall_feature_importance_lasso['SEVERE'].
 ↪tolist()},
                  index=overall_feature_importance_lasso['Feature'].tolist())
haz_severe = haz_severe[haz_severe.SEVERE != 0]
haz_df.sort_values(by=['SEVERE'], ascending=False, inplace=True)

ax = haz_severe.plot.barh(rot=50, figsize=(12, 12), color='green')
ax.set_xlim([0.5,1.5])
ax.grid()
ax.set_title('Hazards From Lasso Model')
fig = ax.get_figure()

fig.savefig('graphs/hazards_lasso_severe.png', bbox_inches='tight')
```



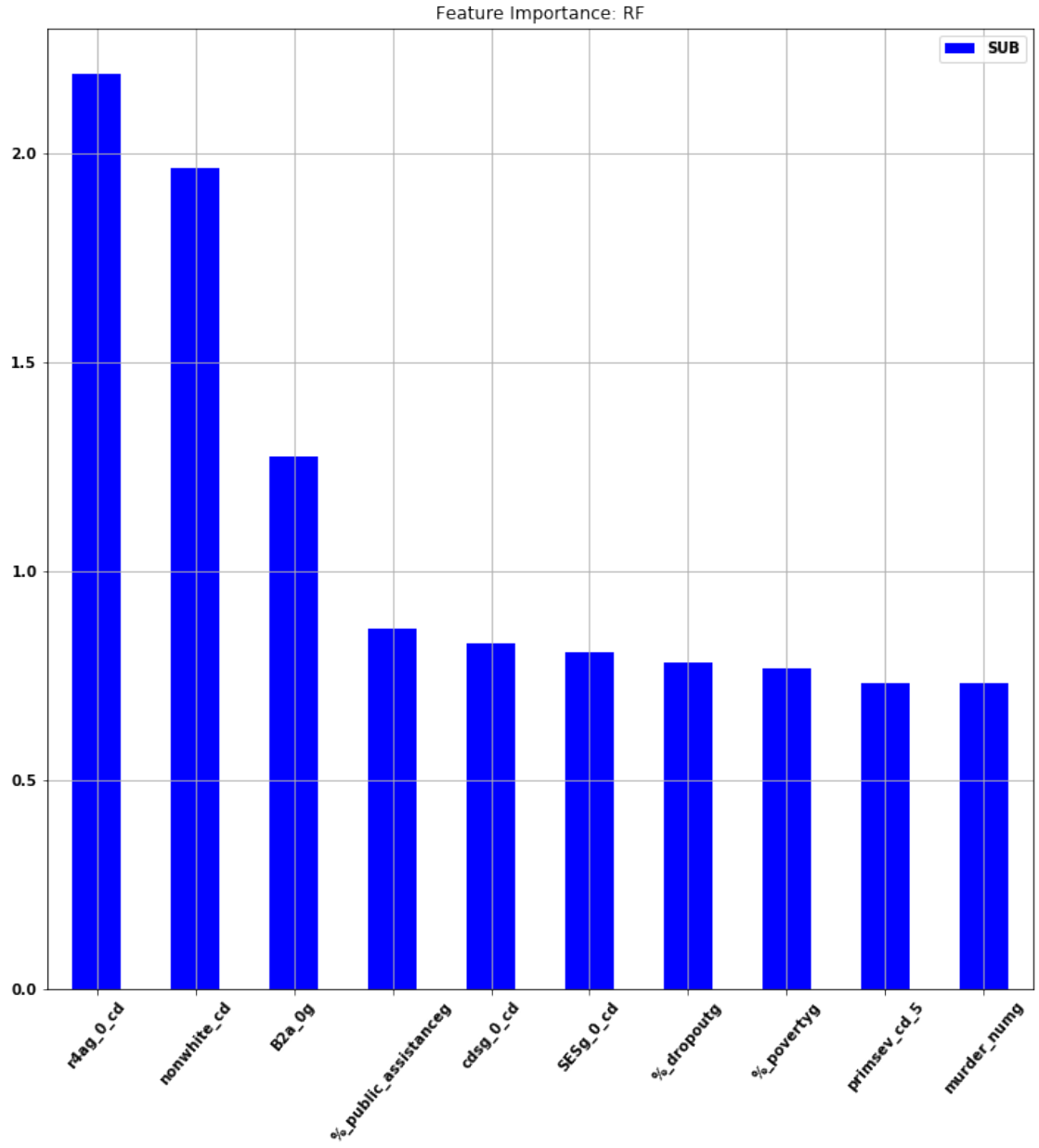
[33]:
```
overall_feature_importance_rf = pd.merge(subclinical_feature_importance_rf,␣
↪mild_feature_importance_rf, on='Feature', how='outer')
overall_feature_importance_rf = pd.merge(overall_feature_importance_rf,␣
↪severe_feature_importance_rf, on='Feature', how='outer')
overall_feature_importance_rf.fillna(0, inplace=True)
display_side_by_side(overall_feature_importance_rf, 4)
```

[34]:
```
# feature importance for rf across all ages
feature_importance = pd.DataFrame({'SUB': overall_feature_importance_rf['SUB'].
↪tolist(),
                   'MILD': overall_feature_importance_rf['MILD'].tolist(),
                   'SEVERE': overall_feature_importance_rf['SEVERE'].tolist()},
                  index=overall_feature_importance_rf['Feature'].tolist())
# John asked to sort this graph by MILD
```

```
feature_importance.sort_values(by=['MILD','SUB','SEVERE'], ascending=False,␣
 ↪inplace=True)
ax = feature_importance.plot.bar(rot=50, figsize=(12, 12))
ax.grid()
ax.set_title('Feature Importance: RF')
fig = ax.get_figure()

fig.savefig('graphs/feature_importance.png', bbox_inches='tight')
```

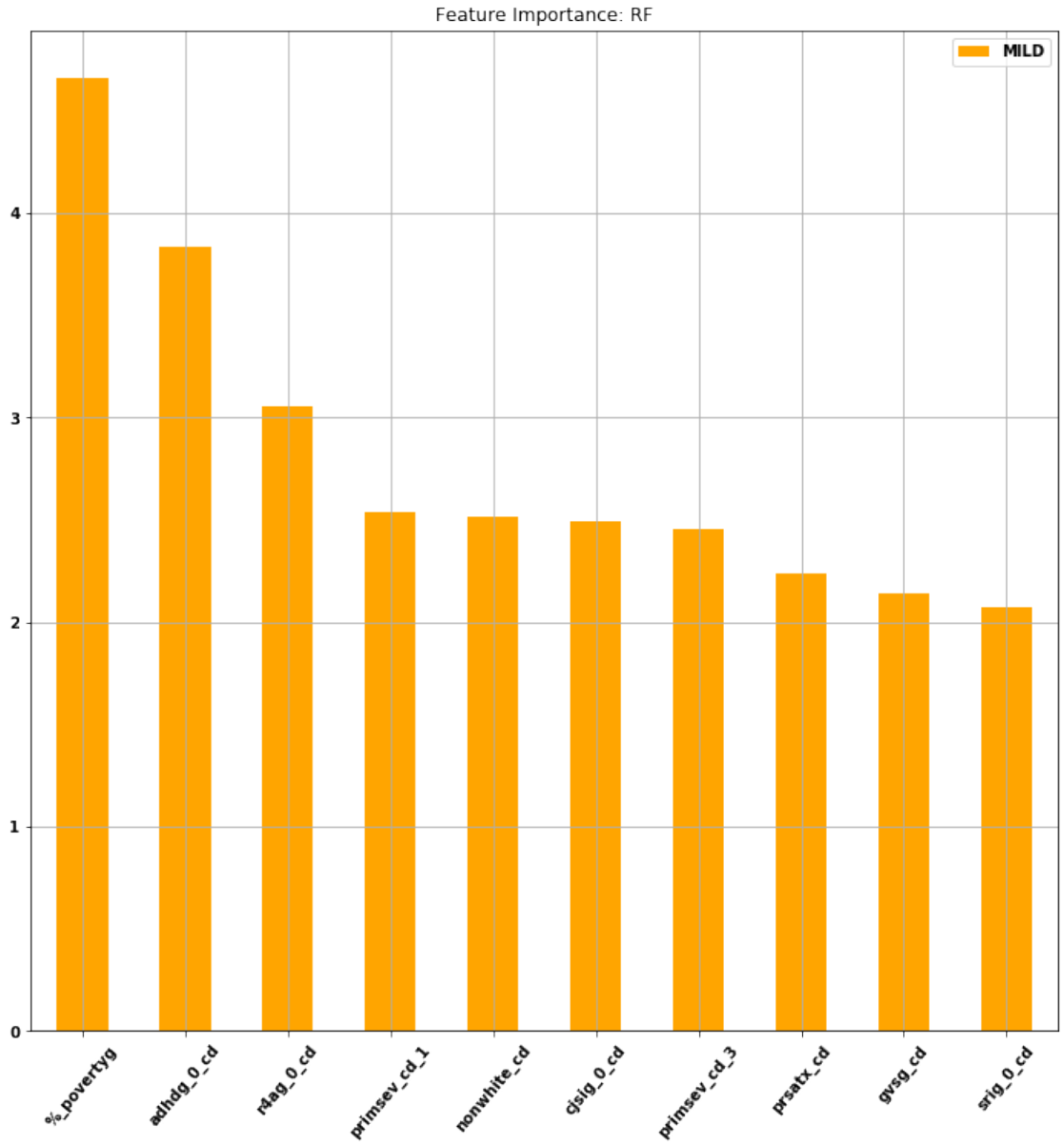


[35]:
```
# feature importance for rf across all ages
feature_importance_sub = pd.DataFrame({'SUB':␣
 ↪overall_feature_importance_rf['SUB'].tolist()},
                  index=overall_feature_importance_rf['Feature'].tolist())
feature_importance_sub = feature_importance_sub[feature_importance_sub.SUB != 0]
# John asked to sort this graph by MILD
feature_importance_sub.sort_values(by=['SUB'], ascending=False, inplace=True)
ax = feature_importance_sub.plot.bar(rot=50, figsize=(12, 12), color='blue')
ax.grid()
ax.set_title('Feature Importance: RF')
fig = ax.get_figure()

fig.savefig('graphs/feature_importance_sub.png', bbox_inches='tight')
```



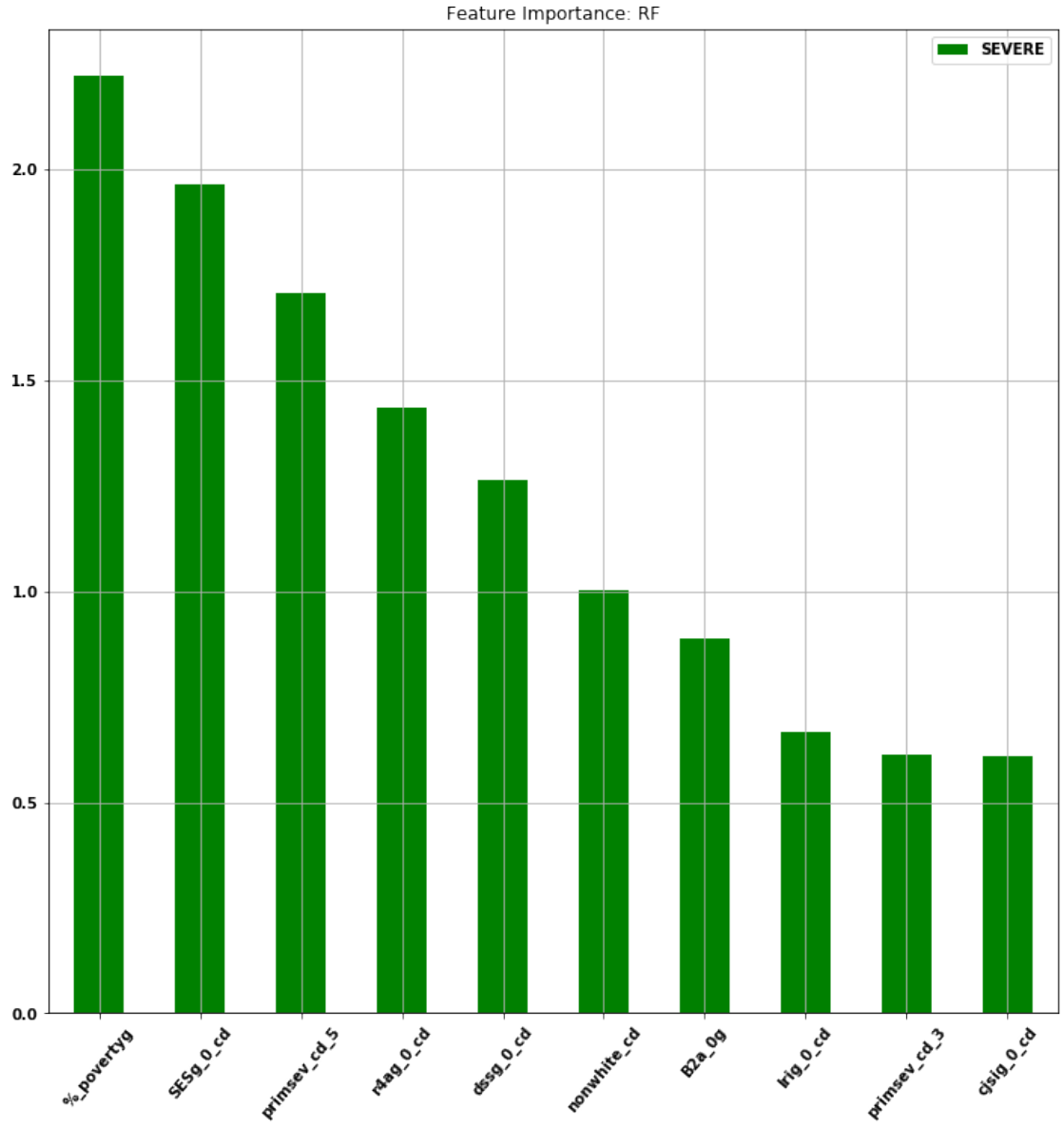
[36]:
```
# feature importance for rf across all ages
feature_importance_mild = pd.DataFrame({'MILD':␣
 ↪overall_feature_importance_rf['MILD'].tolist()},
                  index=overall_feature_importance_rf['Feature'].tolist())
feature_importance_mild = feature_importance_mild[feature_importance_mild.MILD !
 ↪= 0]
# John asked to sort this graph by MILD
feature_importance_mild.sort_values(by=['MILD'], ascending=False, inplace=True)
ax = feature_importance_mild.plot.bar(rot=50, figsize=(12, 12), color='orange')
```

```
ax.grid()
ax.set_title('Feature Importance: RF')
fig = ax.get_figure()

fig.savefig('graphs/feature_importance_mild.png', bbox_inches='tight')
```



[37]:
```
# feature importance for rf across all ages
feature_importance_severe = pd.DataFrame({'SEVERE':␣
↪overall_feature_importance_rf['SEVERE'].tolist()},
                  index=overall_feature_importance_rf['Feature'].tolist())
```

```
feature_importance_severe = feature_importance_severe[feature_importance_severe.
 ↪SEVERE != 0]
# John asked to sort this graph by MILD
feature_importance_severe.sort_values(by=['SEVERE'], ascending=False,␣
 ↪inplace=True)
ax = feature_importance_severe.plot.bar(rot=50, figsize=(12, 12), color='green')
ax.grid()
ax.set_title('Feature Importance: RF')
fig = ax.get_figure()

fig.savefig('graphs/feature_importance_severe.png', bbox_inches='tight')
```



[38]:
```
# top features in both models across all severity groups
rf = overall_feature_importance_rf['Feature'].tolist()
lasso = overall_feature_importance_lasso['Feature'].tolist()

common_features = []
lasso_common_features = []

for elem_rf in rf:
    for elem_lasso in lasso:
        if elem_lasso.startswith(elem_rf):
            common_features.append(elem_rf)
            lasso_common_features.append(elem_lasso)

common_features = list(set(common_features))
lasso_common_features = list(set(lasso_common_features))
```

[39]:
```
def analyze_common_features(subgroup):
    global common_features
    global lasso_common_features

    data = []

    for feat in common_features:
        temp = []
        temp.append(feat)
        temp.append(float(overall_feature_importance_rf.
 ↪loc[overall_feature_importance_rf['Feature'] == feat, subgroup]))
        for i,row in overall_feature_importance_lasso.iterrows():
            if row['Feature'].startswith(feat):
                temp.append((row['Feature'], row[subgroup]))
        data.append(temp)

    df = pd.DataFrame(data=data, columns=['Feature', 'RF', 'Lasso', 'Lasso_'])
    df = df.fillna(0)

    """for i,row in df.iterrows():
        if row['Lasso'][1] == 0:
            df.iloc[i, df.columns.get_loc('Lasso')] = np.nan
    df = df.dropna()


    df = df[df.RF != 0]"""
    return df
```

[40]:
```
analyze_common_features('SUB')
```

[40]:

	Feature	RF	Lasso	Lasso_
0	murder_numg	0.729976	(murder_numg_1, 0.780956774533718)	0
1	r4ag_0_cd	2.188743	(r4ag_0_cd_2, 1.114067715857281)	(r4ag_0_cd_1, 0.0)
2	primsev_cd_5	0.731026	(primsev_cd_5, 0.0)	0
3	prsatx_cd	0.000000	(prsatx_cd_1, 1.0605012361421318)	0
4	cdsg_0_cd	0.826758	(cdsg_0_cd_1, 1.0507292914388484)	0
5	B2a_0g	1.272002	(B2a_0g_2, 1.215113755832353)	(cdsg_0_cd_2, 0.0)
6	adhdg_0_cd	0.000000	(adhdg_0_cd_2, 0.0)	0
7	cjsig_0_cd	0.000000	(cjsig_0_cd_2, 1.3180198588026577)	0
8	gvsg_cd	0.000000	(gvsg_cd_2, 1.0820186888307373)	0
9	srig_0_cd	0.000000	(srig_0_cd_1, 0.0)	(srig_0_cd_2, 0.0)
10	SESg_0_cd	0.803685	(SESg_0_cd_1, 1.0973043072637512)	(SESg_0_cd_2, 0.0)
11	dssg_0_cd	0.000000	(dssg_0_cd_1, 0.0)	(dssg_0_cd_2, 0.0)
12	%_povertyg	0.765828	(%_povertyg_1, 1.1198947285381282)	0
13	nonwhite_cd	1.964530	(nonwhite_cd, 0.0)	0

[41]:
```
analyze_common_features('MILD')
```

[41]:

	Feature	RF	Lasso	Lasso_
0	murder_numg	0.000000	(murder_numg_1, 0.8867009428036636)	0
1	r4ag_0_cd	3.055471	(r4ag_0_cd_2, 1.1474929411917152)	(r4ag_0_cd_1, 0.0)
2	primsev_cd_5	0.000000	(primsev_cd_5, 0.0)	0
3	prsatx_cd	2.235674	(prsatx_cd_1, 1.1842846566918055)	0
4	cdsg_0_cd	0.000000	(cdsg_0_cd_1, 1.061311198212194)	0
5	B2a_0g	0.000000	(B2a_0g_2, 0.0)	(cdsg_0_cd_2, 0.0)
6	adhdg_0_cd	3.838315	(adhdg_0_cd_2, 0.0)	0
7	cjsig_0_cd	2.493893	(cjsig_0_cd_2, 0.0)	0
8	gvsg_cd	2.139436	(gvsg_cd_2, 0.0)	0
9	srig_0_cd	2.073801	(srig_0_cd_1, 0.9084894188799633)	(srig_0_cd_2, 0.0)
10	SESg_0_cd	0.000000	(SESg_0_cd_1, 1.0487247309510852)	0
11	dssg_0_cd	0.000000	(dssg_0_cd_1, 1.0652046554469212)	(dssg_0_cd_2, 0.0)

```
12    %_povertyg  4.659680   (%_povertyg_1, 1.2333772100026164)
13   nonwhite_cd  2.515367    (nonwhite_cd, 0.9148447974743265)

                Lasso_
0                    0
1   (r4ag_0_cd_1, 0.0)
2                    0
3                    0
4   (cdsg_0_cd_2, 0.0)
5                    0
6                    0
7                    0
8                    0
9   (srig_0_cd_2, 0.0)
10  (SESg_0_cd_2, 0.0)
11  (dssg_0_cd_2, 0.0)
12                   0
13                   0
```

[42]:
```
analyze_common_features('SEVERE')
```

[42]:
```
          Feature        RF                                Lasso  \
0    murder_numg  0.000000               (murder_numg_1, 0.0)
1      r4ag_0_cd  1.435621  (r4ag_0_cd_2, 1.1166483894687371)
2   primsev_cd_5  1.707724  (primsev_cd_5, 1.246818506222665)
3      prsatx_cd  0.000000                 (prsatx_cd_1, 0.0)
4      cdsg_0_cd  0.000000                 (cdsg_0_cd_1, 0.0)
5         B2a_0g  0.888512                    (B2a_0g_2, 0.0)
6     adhdg_0_cd  0.000000  (adhdg_0_cd_2, 1.098118622162022)
7     cjsig_0_cd  0.610793                (cjsig_0_cd_2, 0.0)
8        gvsg_cd  0.000000                   (gvsg_cd_2, 0.0)
9      srig_0_cd  0.000000                 (srig_0_cd_1, 0.0)
10     SESg_0_cd  1.962828   (SESg_0_cd_1, 1.048475861016479)
11     dssg_0_cd  1.263581                 (dssg_0_cd_1, 0.0)
12    %_povertyg  2.221177                (%_povertyg_1, 0.0)
13   nonwhite_cd  1.001911                 (nonwhite_cd, 0.0)

                                Lasso_
0                                    0
1   (r4ag_0_cd_1, 1.0862045097784037)
2                                    0
3                                    0
4   (cdsg_0_cd_2, 1.0689662918434981)
5                                    0
6                                    0
7                                    0
8                                    0
```

```
9   (srig_0_cd_2, 1.0655352345930664)
10  (SESg_0_cd_2, 1.0723014790596557)
11  (dssg_0_cd_2, 1.1137352916527563)
12                                  0
13                                  0
```

[43]:
```
# print out total notebook execution time
total_seconds = int(time.time() - start_time)
minutes = total_seconds // 60
seconds = total_seconds % 60
print("--- " + str(minutes) + " minutes " + str(seconds) + " seconds ---")
```

```
--- 10 minutes 41 seconds ---
```

[ ]:
```

```