In [1]:
```
# Load packages
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import sklearn.linear_model as skl
from pandas import concat
from matplotlib.pyplot import subplots
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LassoCV, lasso_path, RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

```
/Users/edithsimochemo/opt/anaconda3/lib/python3.9/site-packages/panda
s/core/computation/expressions.py:21: UserWarning: Pandas requires ve
rsion '2.8.4' or newer of 'numexpr' (version '2.7.3' currently instal
led).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
/Users/edithsimochemo/opt/anaconda3/lib/python3.9/site-packages/panda
s/core/arrays/masked.py:60: UserWarning: Pandas requires version '1.
3.6' or newer of 'bottleneck' (version '1.3.2' currently installed).
  from pandas.core import (
```

In [2]:
```
############################################ DATA MANIPULATION ########
```

In [3]:
```
# Import dataset
data = pd.read_stata("final_fl2.dta")

# Convert Categorical variables: List of categorical columns
categorical_cols = ['race','sex','custody_description','county1']

# Create an instance of LabelEncoder
le = LabelEncoder()

# Apply LabelEncoder to each categorical column
for col in categorical_cols:
    data[col+ '_encoded'] = le.fit_transform(data[col])

# Select columns
data = data.select_dtypes(include=['number'])

# Removing missing observations
data = data.dropna()
data.shape
```

Out[3]:
```
(129531, 90)
```

In [4]:
```
# Define variables
X = data.drop(columns = ["anyrecid","distn3","distn4","finrecidany","r
Y = data["anyrecid"]

# Print the column heads
print(X.columns)
```

```
Index(['level_0', 'index', 'date', 'adate', 'rdate', 'releaseyear',
       'releasemonth', 'after', 'dist', 'distnoab', 'distn2', 'fullba
nafter',
       'fullbanbefore', 'concurrent_sentence', 'drugoffense', 'traffo
ffense',
       'otheroffense', 'smd', 'traffmar', 'traffcoc', 'traffher', 'tr
affamph',
       'traffill', 'traffconspir', 'fincrime', 'notpossoffense',
       'drugoffense_noselling', 'drugoffense_poss', 'violentcrime', '
assault',
       'elderly', 'escape', 'forgery', 'fraud', 'kidnap', 'manslaught
er',
       'murder', 'othercrime', 'otherviolent', 'propdamage', 'rackete
er',
       'robbery', 'sexcrime', 'propsteal', 'weapon', 'criminalmischie
f', 'dui',
       'licrevoke', 'fleeorescape', 'fraudforge', 'anytheft', 'anybur
g',
       'propcrime', 'birthyear', 'maxdate', 'maxadate', 'maxrdate', '
dateorig',
       'offenseyear', 'offensemonth', 'ban', 'age', 'under30', 'blac
k', 'male',
       'totalyearssentenced', 'prioroffensenumber', 'prioroffense',
       'countoffenses', 'preoct97', 'placebodrug', 'placebosmd', 'yea
r_x',
       'month_x', 'unemp_rate', 'year_y', 'lincome', 'year', 'month_
y',
       'percentage_snap_recipients', 'race_encoded', 'sex_encoded',
       'custody_description_encoded', 'county1_encoded'],
      dtype='object')
```

In [5]:
```
######################################## OLS REGRESSION #############
```

In [6]:
```
# Fit OLS
model = sm.OLS(Y,X).fit()
```

In [7]:
```
# Latex Output
latex_output = model.summary().as_latex()
print(latex_output)
```

```
\begin{center}
```

```
\begin{tabular}{lclc}
\toprule
\textbf{Dep. Variable:}                & anyrecid         & \textbf{
R-squared:          } &     0.201   \\
\textbf{Model:}                        &       OLS        & \textbf{
Adj. R-squared:     } &     0.200   \\
\textbf{Method:}                       &  Least Squares   & \textbf{
F-statistic:        } &     570.2   \\
\textbf{Date:}                         & Fri, 20 Dec 2024 & \textbf{
Prob (F-statistic):} &     0.00    \\
\textbf{Time:}                         &     20:01:08     & \textbf{
Log-Likelihood:     } &   -75160.   \\
\textbf{No. Observations:}             &      129531      & \textbf{
AIC:                } & 1.504e+05   \\
\textbf{Df Residuals:}                 &      129473      & \textbf{
BIC:                } & 1.510e+05   \\
\textbf{Df Model:}                     &          57      & \textbf{
} &             \\
\textbf{Covariance Type:}              &    nonrobust     & \textbf{
} &             \\
\bottomrule
\end{tabular}
\begin{tabular}{lcccccc}
                                       & \textbf{coef} & \textbf{std
err} & \textbf{t} & \textbf{P$> |$t$|$} & \textbf{[0.025} & \textbf{
0.975]}  \\
\midrule
\textbf{level\_0}                      &   -3.898e-09  &    4.53e-09
&    -0.860  &        0.390        &    -1.28e-08    &    4.99e-09
\\
\textbf{index}                         &   -2.268e-09  &    4.51e-09
&    -0.503  &        0.615        &    -1.11e-08    &    6.57e-09
\\
\textbf{date}                          &       1.0942  &       0.205
&     5.340  &        0.000        &        0.693    &       1.496
\\
\textbf{adate}                         &   -7.004e-05  &    1.44e-05
&    -4.872  &        0.000        &    -9.82e-05    &   -4.19e-05
\\
\textbf{rdate}                         &       0.8377  &       0.155
&     5.397  &        0.000        &        0.533    &       1.142
\\
\textbf{releaseyear}                   &       0.8005  &       0.142
&     5.631  &        0.000        &        0.522    &       1.079
\\
\textbf{releasemonth}                  &       0.0056  &       0.001
&     4.152  &        0.000        &        0.003    &       0.008
\\
\textbf{after}                         &       0.0057  &       0.014
&     0.412  &        0.681        &       -0.021    &       0.033
```

```
\\
\textbf{dist}                         &    2.054e-06  &     3.67e-05
&     0.056  &         0.955        &    -6.99e-05    &     7.4e-05
\\
\textbf{distnoab}                     &       0.0235  &        0.010
&     2.349  &         0.019        &       0.004    &       0.043
\\
\textbf{distn2}                       &   -8.911e-09  &     5.39e-10
&   -16.528  &         0.000        &    -9.97e-09    &    -7.85e-09
\\
\textbf{fullbanafter}                 &       0.0001  &     2.43e-05
&     5.399  &         0.000        &     8.37e-05    &       0.000
\\
\textbf{fullbanbefore}                &    -4.84e-05  &     8.96e-06
&    -5.399  &         0.000        &     -6.6e-05    &    -3.08e-05
\\
\textbf{concurrent\_sentence}         &    -367.3576  &       68.045
&    -5.399  &         0.000        &     -500.725    &     -233.990
\\
\textbf{drugoffense}                  &       0.0021  &        0.000
&     5.392  &         0.000        &       0.001    &       0.003
\\
\textbf{traffoffense}                 &    -1.13e-05  &     2.09e-06
&    -5.399  &         0.000        &    -1.54e-05    &     -7.2e-06
\\
\textbf{otheroffense}                 &       0.0021  &        0.000
&     5.392  &         0.000        &       0.001    &       0.003
\\
\textbf{smd}                          &       0.0016  &        0.006
&     0.268  &         0.788        &      -0.010    &       0.014
\\
\textbf{traffmar}                     &    3.645e-06  &     6.75e-07
&     5.399  &         0.000        &     2.32e-06    &     4.97e-06
\\
\textbf{traffcoc}                     &    4.866e-06  &     9.01e-07
&     5.399  &         0.000        &      3.1e-06    &     6.63e-06
\\
\textbf{traffher}                     &   -8.549e-06  &     1.58e-06
&    -5.399  &         0.000        &    -1.17e-05    &    -5.45e-06
\\
\textbf{traffamph}                    &     7.69e-06  &     1.42e-06
&     5.399  &         0.000        &      4.9e-06    &     1.05e-05
\\
\textbf{traffill}                     &   -3.293e-06  &      6.1e-07
&    -5.399  &         0.000        &    -4.49e-06    &     -2.1e-06
\\
\textbf{traffconspir}                 &   -9.527e-06  &     1.76e-06
&    -5.399  &         0.000        &     -1.3e-05    &    -6.07e-06
\\
\textbf{fincrime}                     &       0.0200  &        0.006
```

```
& 3.346 & 0.001 & 0.008 & 0.032
\\
\textbf{notpossoffense} & -0.0059 & 0.012
& -0.509 & 0.611 & -0.028 & 0.017
\\
\textbf{drugoffense\_noselling} & 0.0246 & 0.013
& 1.907 & 0.056 & -0.001 & 0.050
\\
\textbf{drugoffense\_poss} & 0.0135 & 0.013
& 1.050 & 0.294 & -0.012 & 0.039
\\
\textbf{violentcrime} & 0.0088 & 0.010
& 0.906 & 0.365 & -0.010 & 0.028
\\
\textbf{assault} & 0.0020 & 0.009
& 0.219 & 0.826 & -0.016 & 0.020
\\
\textbf{elderly} & 0.0198 & 0.025
& 0.806 & 0.420 & -0.028 & 0.068
\\
\textbf{escape} & 0.0128 & 0.015
& 0.833 & 0.405 & -0.017 & 0.043
\\
\textbf{forgery} & 0.0086 & 0.012
& 0.691 & 0.490 & -0.016 & 0.033
\\
\textbf{fraud} & -0.0190 & 0.024
& -0.795 & 0.427 & -0.066 & 0.028
\\
\textbf{kidnap} & -0.0368 & 0.021
& -1.764 & 0.078 & -0.078 & 0.004
\\
\textbf{manslaughter} & -0.0470 & 0.041
& -1.153 & 0.249 & -0.127 & 0.033
\\
\textbf{murder} & -0.0680 & 0.033
& -2.070 & 0.038 & -0.132 & -0.004
\\
\textbf{othercrime} & 0.0201 & 0.006
& 3.099 & 0.002 & 0.007 & 0.033
\\
\textbf{otherviolent} & -0.0306 & 0.014
& -2.218 & 0.027 & -0.058 & -0.004
\\
\textbf{propdamage} & 0.0471 & 0.038
& 1.255 & 0.209 & -0.026 & 0.121
\\
\textbf{racketeer} & -0.0620 & 0.044
& -1.409 & 0.159 & -0.148 & 0.024
\\
```

```
\textbf{robbery}                     &       0.0151  &        0.013
&     1.126  &         0.260         &       -0.011    &       0.041
\\
\textbf{sexcrime}                    &       0.0688  &        0.023
&     3.033  &         0.002         &        0.024    &       0.113
\\
\textbf{propsteal}                   &       0.0667  &        0.015
&     4.383  &         0.000         &        0.037    &       0.097
\\
\textbf{weapon}                      &      -0.0216  &        0.007
&    -3.143  &         0.002         &       -0.035    &      -0.008
\\
\textbf{criminalmischief}            &      -0.0548  &        0.041
&    -1.342  &         0.180         &       -0.135    &       0.025
\\
\textbf{dui}                         &      -0.0175  &        0.019
&    -0.899  &         0.369         &       -0.056    &       0.021
\\
\textbf{licrevoke}                   &       0.0268  &        0.008
&     3.402  &         0.001         &        0.011    &       0.042
\\
\textbf{fleeorescape}                &       0.0373  &        0.007
&     5.048  &         0.000         &        0.023    &       0.052
\\
\textbf{fraudforge}                  &       0.0249  &        0.012
&     2.145  &         0.032         &        0.002    &       0.048
\\
\textbf{anytheft}                    &       0.0179  &        0.007
&     2.593  &         0.010         &        0.004    &       0.031
\\
\textbf{anyburg}                     &      -0.0030  &        0.006
&    -0.470  &         0.638         &       -0.015    &       0.009
\\
\textbf{propcrime}                   &      -0.0297  &        0.015
&    -1.929  &         0.054         &       -0.060    &       0.000
\\
\textbf{birthyear}                   &       0.0074  &        0.004
&     1.666  &         0.096         &       -0.001    &       0.016
\\
\textbf{maxdate}                     &    1.756e-05  &     4.56e-06
&     3.848  &         0.000         &     8.62e-06    &    2.65e-05
\\
\textbf{maxadate}                    &   -4.127e-05  &     5.38e-06
&    -7.667  &         0.000         &    -5.18e-05    &   -3.07e-05
\\
\textbf{maxrdate}                    &      -0.8381  &        0.155
&    -5.400  &         0.000         &       -1.142    &      -0.534
\\
\textbf{dateorig}                    &      -1.1177  &        0.205
&    -5.454  &         0.000         &       -1.519    &      -0.716
```

```
\\
\textbf{offenseyear}                   &       0.0105  &       0.050
&     0.211  &        0.833        &      -0.087    &       0.108
\\
\textbf{offensemonth}                  &       0.0013  &       0.004
&     0.317  &        0.751        &      -0.007    &       0.009
\\
\textbf{ban}                           &            0  &           0
&       nan  &          nan        &           0    &           0
\\
\textbf{age}                           &      -0.0020  &       0.004
&    -0.461  &        0.645        &      -0.011    &       0.007
\\
\textbf{under30}                       &       0.0129  &       0.004
&     3.187  &        0.001        &       0.005    &       0.021
\\
\textbf{black}                         &       0.0330  &       0.051
&     0.646  &        0.518        &      -0.067    &       0.133
\\
\textbf{male}                          &       0.0348  &       0.002
&    18.796  &        0.000        &       0.031    &       0.038
\\
\textbf{totalyearssentenced}           &    1.341e+05  &    2.48e+04
&     5.399  &        0.000        &    8.54e+04    &    1.83e+05
\\
\textbf{prioroffensenumber}            &       0.0492  &       0.001
&    37.604  &        0.000        &       0.047    &       0.052
\\
\textbf{prioroffense}                  &       0.0730  &       0.004
&    20.341  &        0.000        &       0.066    &       0.080
\\
\textbf{countoffenses}                 &       0.0008  &       0.005
&     0.184  &        0.854        &      -0.008    &       0.010
\\
\textbf{preoct97}                      &            0  &           0
&       nan  &          nan        &           0    &           0
\\
\textbf{placebodrug}                   &       0.0021  &       0.000
&     5.392  &        0.000        &       0.001    &       0.003
\\
\textbf{placebosmd}                    &       0.0016  &       0.006
&     0.267  &        0.789        &      -0.010    &       0.014
\\
\textbf{year\_x}                       &       0.4748  &       0.082
&     5.778  &        0.000        &       0.314    &       0.636
\\
\textbf{month\_x}                      &       0.0032  &       0.001
&     2.427  &        0.015        &       0.001    &       0.006
\\
\textbf{unemp\_rate}                   &       0.0052  &       0.001
```

```
& 7.368 & 0.000 & 0.004 & 0.007
\\
\textbf{year\_y} & -1.6064 & 0.304
& -5.278 & 0.000 & -2.203 & -1.010
\\
\textbf{lincome} & -0.0535 & 0.011
& -5.084 & 0.000 & -0.074 & -0.033
\\
\textbf{year} & 0.4748 & 0.082
& 5.778 & 0.000 & 0.314 & 0.636
\\
\textbf{month\_y} & 0.0032 & 0.001
& 2.427 & 0.015 & 0.001 & 0.006
\\
\textbf{percentage\_snap\_recipients} & -0.3606 & 0.037
& -9.621 & 0.000 & -0.434 & -0.287
\\
\textbf{race\_encoded} & -0.0047 & 0.017
& -0.275 & 0.783 & -0.038 & 0.029
\\
\textbf{sex\_encoded} & 0.0350 & 0.002
& 18.915 & 0.000 & 0.031 & 0.039
\\
\textbf{custody\_description\_encoded} & 0.0014 & 0.001
& 1.724 & 0.085 & -0.000 & 0.003
\\
\textbf{county1\_encoded} & 0.0002 & 6.06e-05
& 2.495 & 0.013 & 3.25e-05 & 0.000
\\
\bottomrule
\end{tabular}
\begin{tabular}{lclc}
\textbf{Omnibus:} & 109341.029 & \textbf{ Durbin-Watson: }
& 1.992 \\
\textbf{Prob(Omnibus):} & 0.000 & \textbf{ Jarque-Bera (JB): }
& 9480.847 \\
\textbf{Skew:} & 0.289 & \textbf{ Prob(JB): }
& 0.00 \\
\textbf{Kurtosis:} & 1.807 & \textbf{ Cond. No. }
& 1.30e+16 \\
\bottomrule
\end{tabular}
%\caption{OLS Regression Results}
\end{center}

Notes: \newline
 [1] Standard Errors assume that the covariance matrix of the errors
is correctly specified. \newline
 [2] The smallest eigenvalue is 1.92e-13. This might indicate that th
ere are \newline
```

strong multicollinearity problems or that the design matrix is singular.

In [8]:
```
################################## RIDGE REGRESSION ###################
```

In [9]:
```
# Load packages
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error as mse
```

In [10]:
```
# Standardized the variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [11]:
```
import warnings

# Suppress all warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

# Set up a range of lambdas
    lambdas = 10**np.linspace(5, -5, 100) / Y.std()
    soln_array = skl.ElasticNet.path(X_scaled,
                                      Y,
                                      l1_ratio=0,
                                      alphas=lambdas)[1]
    print(soln_array.shape)
```

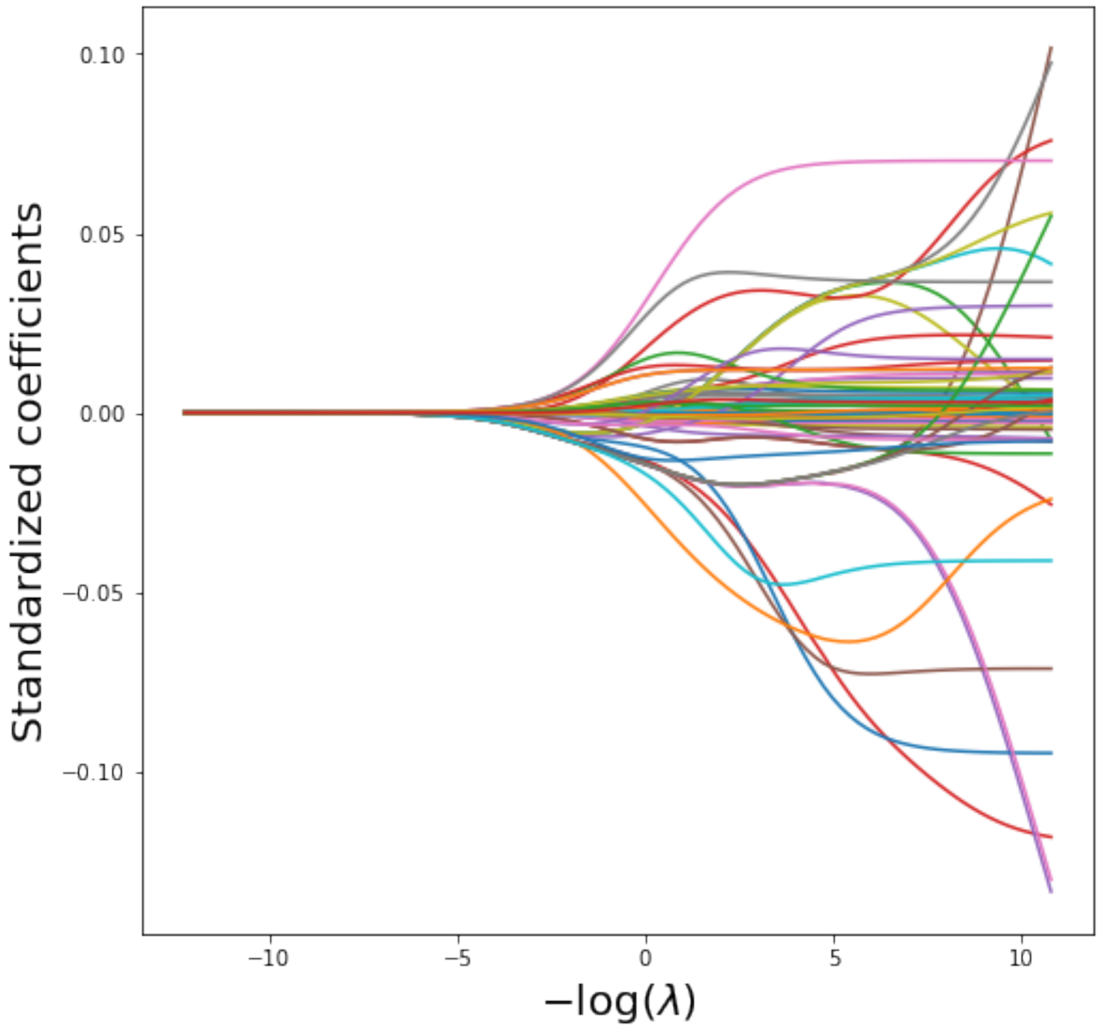
```
(84, 100)
```

In [12]:
```python
# Transform Soln_path into a dataframe
soln_path = pd.DataFrame(soln_array.T,
                         columns=X.columns,
                         index=-np.log(lambdas))
soln_path.index.name = 'negative log(lambda)'
soln_path
```

Out[12]:

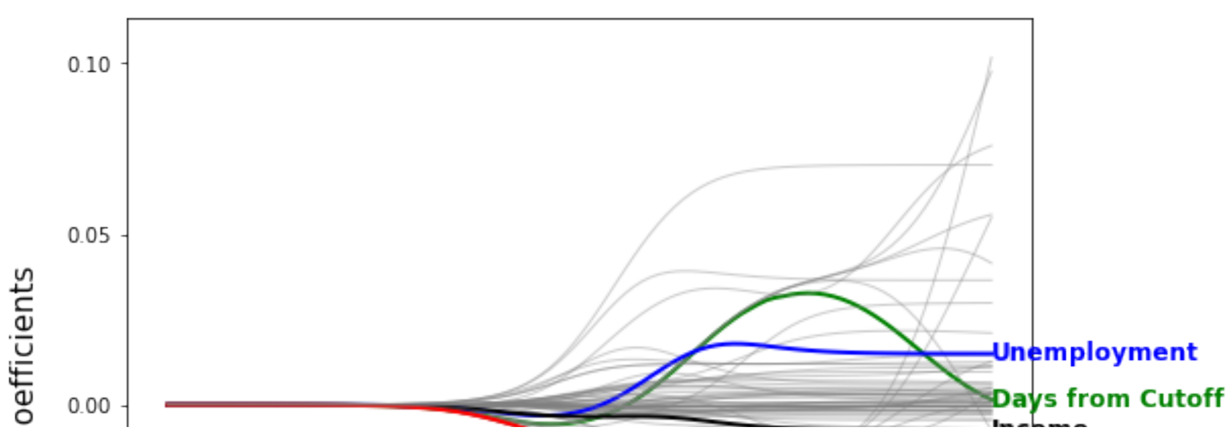
ate	year_y	lincome	year	month_y	percentage_snap_recipients	race_encoded
9e-07	-8.073586e-07	-3.510545e-07	-8.073586e-07	1.855850e-09	-7.588481e-07	-3.429147e-07
8e-07	-1.018752e-06	-4.429712e-07	-1.018752e-06	2.341490e-09	-9.575399e-07	-4.327054e-07
9e-07	-1.285488e-06	-5.589514e-07	-1.285488e-06	2.954104e-09	-1.208250e-06	-5.460063e-07
5e-07	-1.622052e-06	-7.052930e-07	-1.622052e-06	3.726828e-09	-1.524594e-06	-6.889722e-07
9e-06	-2.046719e-06	-8.899411e-07	-2.046719e-06	4.701403e-09	-1.923747e-06	-8.693690e-07
...	...	...	...	...	...	...
1e-02	8.655230e-03	-7.427179e-03	-9.063092e-04	1.257519e-03	-4.118610e-02	-7.989319e-03
0e-02	9.832555e-03	-7.428573e-03	-4.928268e-04	1.397270e-03	-4.117963e-02	-7.953269e-03
0e-02	1.089884e-02	-7.429723e-03	-1.407429e-04	1.524582e-03	-4.117455e-02	-7.921271e-03
5e-02	1.185477e-02	-7.430680e-03	1.562518e-04	1.639241e-03	-4.117054e-02	-7.892679e-03
0e-02	1.270382e-02	-7.431485e-03	4.046866e-04	1.741431e-03	-4.116730e-02	-7.866986e-03

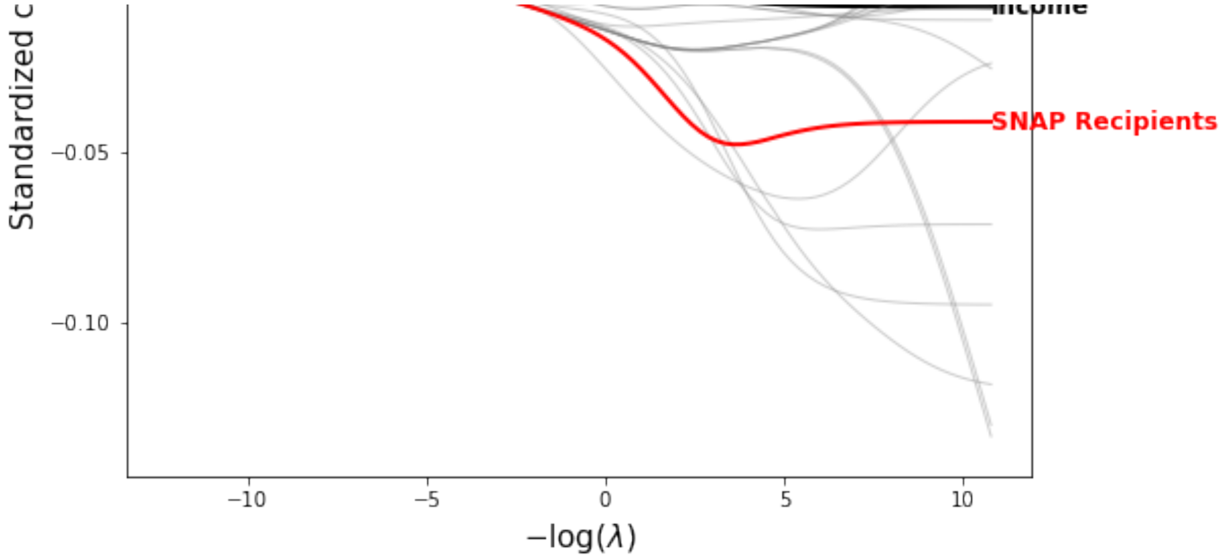
In [13]:
```
# Plot the graph
path_fig, ax = subplots(figsize=(8,8))
soln_path.plot(ax=ax, legend=False)
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Standardized coefficients', fontsize=20)
#ax.legend(loc='upper left', bbox_to_anchor=(1.1, 1.005))
#ax.set_title('Ridge Regression Coefficient Path', fontsize=21);
plt.savefig('ridge_final.png', format = 'png', dpi=300, bbox_inches='t
#ax.set_ylim([-0.1,0.4]);
```



In [14]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import to_rgba

# Plot the graph
fig, ax = plt.subplots(figsize=(8, 8))

# Plot all the coefficient paths
soln_path.plot(ax=ax, legend=False)
```

```
# List of variables you want to highlight with their specific colors a
highlight_vars = {
    'lincome': ('black', 'Income'),
    'unemp_rate': ('blue', 'Unemployment'),
    'dist': ('green', 'Days from Cutoff'),
    'percentage_snap_recipients': ('red', 'SNAP Recipients')
}

# Iterate over the plotted lines
for i, line in enumerate(ax.get_lines()):
    var_name = soln_path.columns[i]

    # Check if the current variable is one to highlight
    if var_name in highlight_vars:
        color, label = highlight_vars[var_name]
        line.set_color(color)
        line.set_linewidth(2)

        # Annotate the line
        x_data = line.get_xdata()[-1]
        y_data = line.get_ydata()[-1]
        ax.text(x_data, y_data, label, color=color, fontsize=12, fontw
                verticalalignment='center', horizontalalignment='left'
    else:
        # Fade the other variables
        line.set_color(to_rgba('gray', alpha=0.4))
        line.set_linewidth(1)

# Set labels and title
ax.set_xlabel('$-\log(\lambda)$', fontsize=15)
ax.set_ylabel('Standardized coefficients', fontsize=15)
#ax.set_title('Ridge Regression Coefficient Path', fontsize=21)
ax.legend().set_visible(False)

# save the figure
plt.savefig('ridge_faded.png', format='png', dpi=300, bbox_inches='tig

plt.show();
```



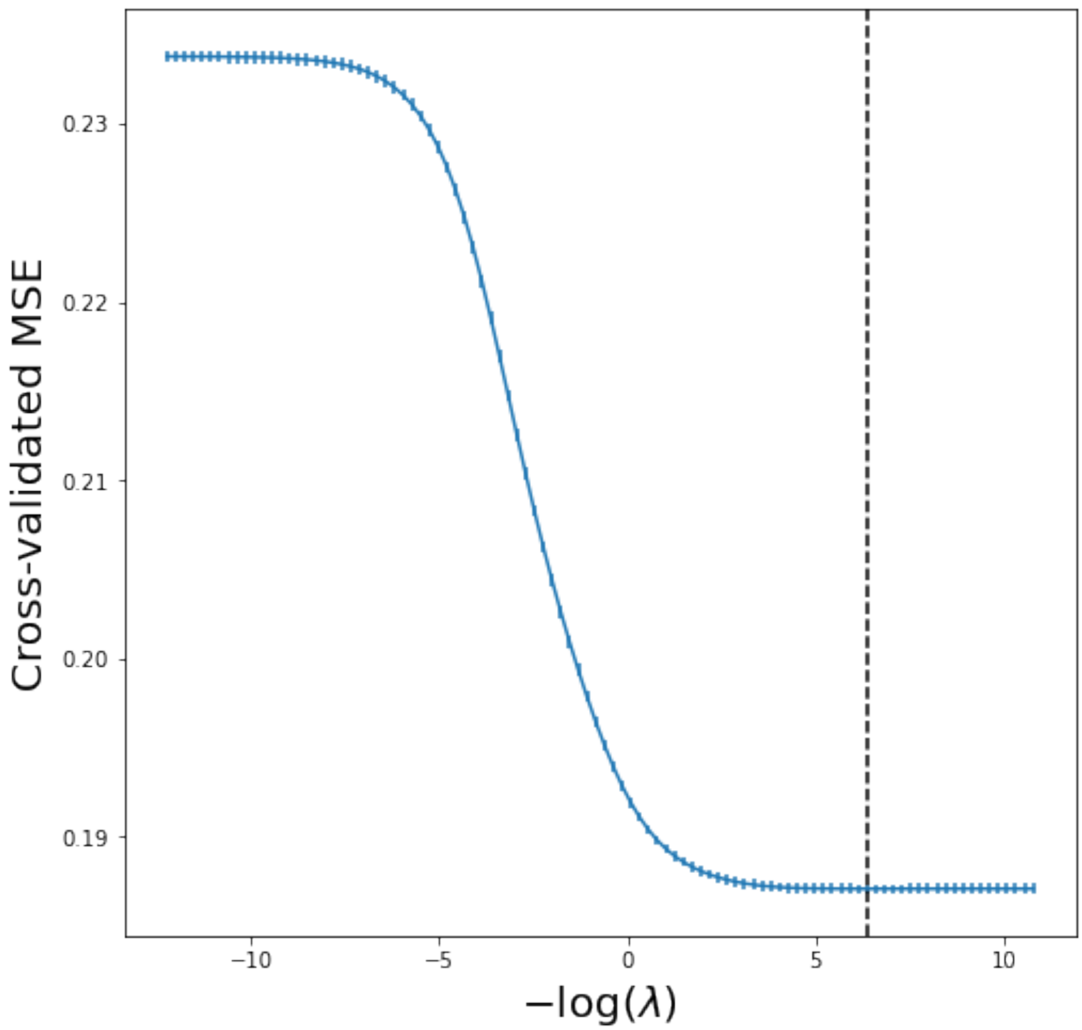


In [15]:
```
# Ridge cross-validation plot
# Set up cross validation
K = 5
kfold = KFold(n_splits = K, random_state=0, shuffle=True)
```

In [16]:
```
import warnings

# Suppress all warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

# Perform RidgeCV with different alpha (lambda) values
    ridgeCV = skl.ElasticNetCV(alphas=lambdas,
                               l1_ratio=0,
                               cv=kfold)
    pipeCV = Pipeline(steps=[('scaler', scaler),
                             ('ridge', ridgeCV)])
    print(pipeCV.fit(X, Y))
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('ridge',
                 ElasticNetCV(alphas=array([2.06826854e+05, 1.6390674
5e+05, 1.29893292e+05, 1.02938213e+05,
       8.15767731e+04, 6.46481976e+04, 5.12325910e+04, 4.06009522e+0
4,
       3.21755603e+04, 2.54985813e+04, 2.02071896e+04, 1.60138522e+0
4,
       1.26907040e+04, 1.00571659e+04, 7.97013196e+03, 6.31619328e+0
3,
       5.00547516e+03, 3.96675346e+03,...
       1.71711288e-03, 1.36078259e-03, 1.07839693e-03, 8.54611126e-0
4,
       6.77264702e-04, 5.36720694e-04, 4.25341971e-04, 3.37076238e-0
4,
       2.67127154e-04, 2.11693701e-04, 1.67763638e-04, 1.32949814e-0
4,
       1.05360454e-04, 8.34963580e-05, 6.61694358e-05, 5.24381462e-0
5,
       4.15563341e-05, 3.29326841e-05, 2.60985890e-05, 2.06826854e-0
5]),
                              cv=KFold(n_splits=5, random_state=0, sh
uffle=True),
                              l1_ratio=0))])
```

In [17]:
```
tuned_ridge = pipeCV.named_steps['ridge']
ridgeCV_fig, ax = subplots(figsize=(8,8))
ax.errorbar(-np.log(lambdas),
            tuned_ridge.mse_path_.mean(1),
            yerr=tuned_ridge.mse_path_.std(1) / np.sqrt(K))
ax.axvline(-np.log(tuned_ridge.alpha_), c='k', ls='--')
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20);
```



In [18]:
```
######################################## LASSO REGRESSION ##############
```

In [19]:
```
# Lasso
import warnings
import matplotlib.pyplot as plt

# Suppress all warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    lassoCV = skl.ElasticNetCV(n_alphas=200, l1_ratio=1, cv=kfold)
    pipeCV = Pipeline(steps=[('scaler', scaler), ('lasso', lassoCV)])
    pipeCV.fit(X, Y)
    tuned_lasso = pipeCV.named_steps['lasso']
    best_alpha = tuned_lasso.alpha_
```

In [20]:
```
# Compute Lasso path
#lambdas, soln_array = lasso_path(X_scaled, Y.values.ravel(), alphas=n

lambdas, soln_array = lasso_path(X_scaled, Y,
                                 alphas=np.logspace(-4, 4, 100),
                                 max_iter=10000)[:2]
soln_path = pd.DataFrame(soln_array.T,
                         columns=X.columns,
                         index=-np.log(lambdas))
soln_path
```

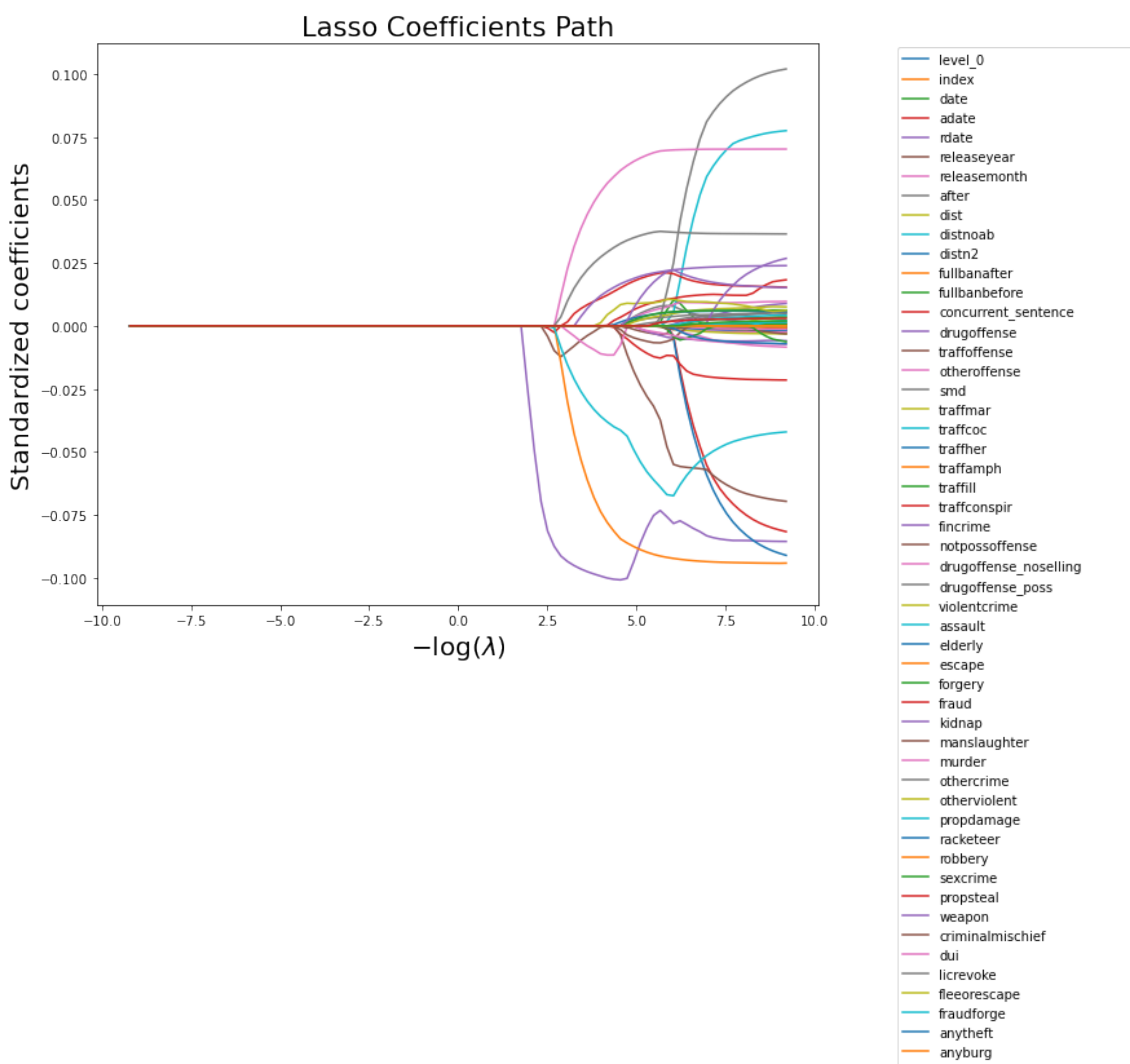
Out[20]:

ab	...	unemp_rate	year_y	lincome	year	month_y	percentage_snap_recipients	race_encoded
)0	...	-0.000000	-0.0	-0.000000	-0.0	0.0	-0.000000	-0.000000
)0	...	-0.000000	-0.0	-0.000000	-0.0	0.0	-0.000000	-0.000000
)0	...	-0.000000	-0.0	-0.000000	-0.0	0.0	-0.000000	-0.000000
)0	...	-0.000000	-0.0	-0.000000	-0.0	0.0	-0.000000	-0.000000
)0	...	-0.000000	-0.0	-0.000000	-0.0	0.0	-0.000000	-0.000000
...	...	...	...	...	...	...	...	...
)7	...	0.015556	-0.0	-0.007064	-0.0	0.0	-0.043357	-0.006769
73	...	0.015444	-0.0	-0.007121	-0.0	0.0	-0.042943	-0.006841
14	...	0.015352	-0.0	-0.007168	-0.0	0.0	-0.042604	-0.006911
42	...	0.015276	-0.0	-0.007207	-0.0	0.0	-0.042324	-0.007021
17	...	0.015210	-0.0	-0.007239	-0.0	0.0	-0.042085	-0.007126

In [21]:
```
# Plot the solution path
```

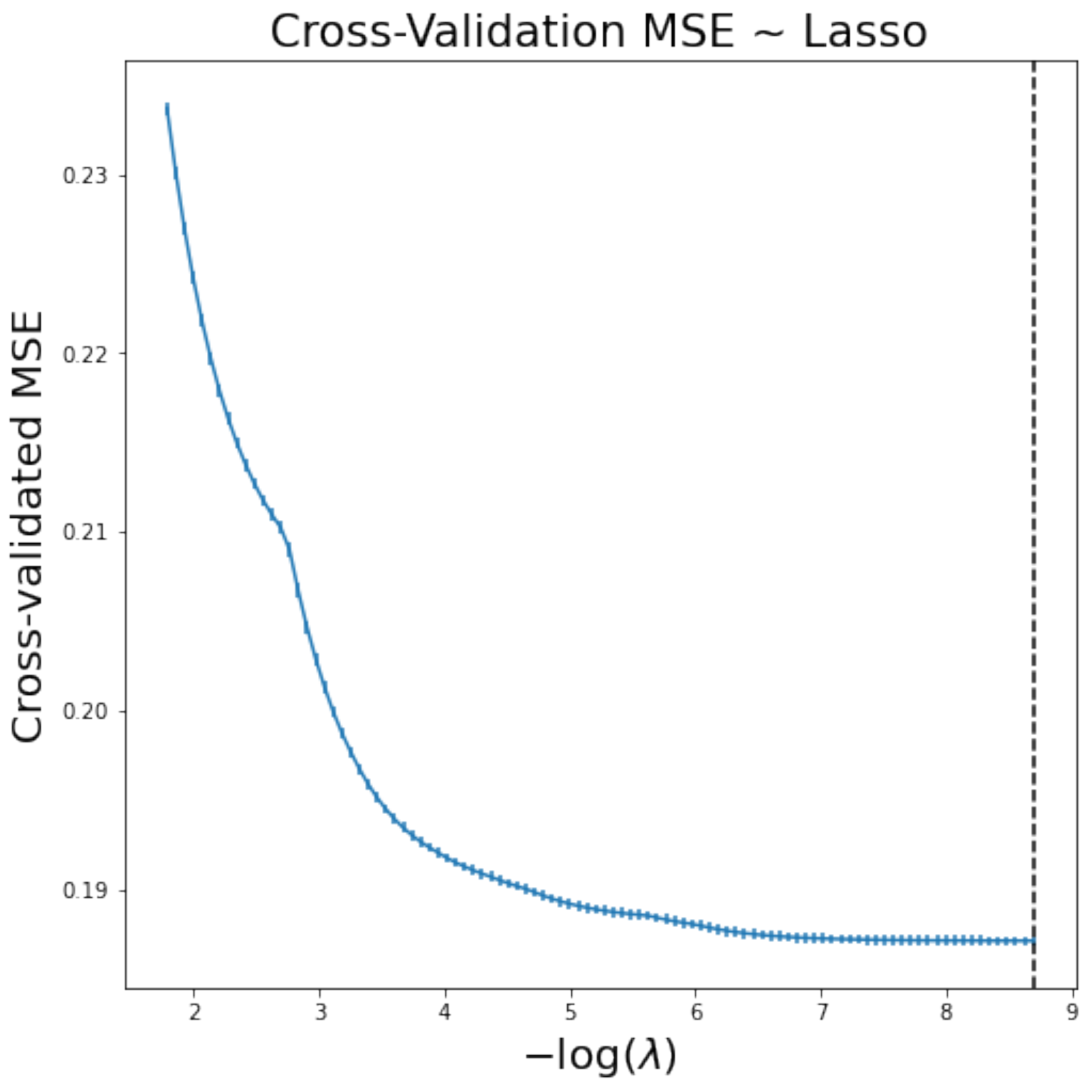
```
plt.figure(figsize=(10, 8))
ax = plt.gca()

for column in soln_path.columns:
    ax.plot(soln_path.index.to_numpy(), soln_path[column].to_numpy(),

ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Standardized coefficients', fontsize=20)
ax.set_title('Lasso Coefficients Path', fontsize=21)
ax.legend(loc='upper left', bbox_to_anchor=(1.1, 1.005))
plt.tight_layout()
plt.show();
```

```
/var/folders/1x/fvyxnz7d5db2mw8jz8k69snr0000gn/T/ipykernel_48306/2188
380166.py:12: UserWarning: Tight layout not applied. The bottom and t
op margins cannot be made large enough to accommodate all axes decora
tions.
  plt.tight_layout()
```



In [22]:
```
# Cross validation MSE
import warnings

# Suppress all warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    lassoCV = skl.ElasticNetCV(n_alphas=100,
                               l1_ratio=1,
                               cv=kfold)
    pipeCV = Pipeline(steps=[('scaler', scaler),
                             ('lasso', lassoCV)])
    pipeCV.fit(X, Y)
    tuned_lasso = pipeCV.named_steps['lasso']
    tuned_lasso.alpha_
```

In [23]:
```
lassoCV_fig, ax = subplots(figsize=(8,8))
ax.errorbar(-np.log(tuned_lasso.alphas_),
            tuned_lasso.mse_path_.mean(1),
            yerr=tuned_lasso.mse_path_.std(1) / np.sqrt(K))
ax.axvline(-np.log(tuned_lasso.alpha_), c='k', ls='--')
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20)
ax.set_title('Cross-Validation MSE ~ Lasso', fontsize =21);
```



In [24]:
```
################################################ CLASSIFICATION TREE ###
```

In [25]:
```
# Load packages
import sklearn.model_selection as skm
from sklearn.tree import (DecisionTreeClassifier as DTC,
                          DecisionTreeRegressor as DTR,
                          plot_tree,
                          export_text)
from sklearn.metrics import (accuracy_score,
                             log_loss)
from sklearn.ensemble import \
     (RandomForestRegressor as RF,
      GradientBoostingRegressor as GBR,
      GradientBoostingClassifier as GBC,
     RandomForestClassifier as RFC)
```

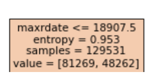
In [26]:
```
# classification Tree
clf = DTC(criterion='entropy',
          max_depth=3,
          random_state=0)
clf.fit(X,Y)
```

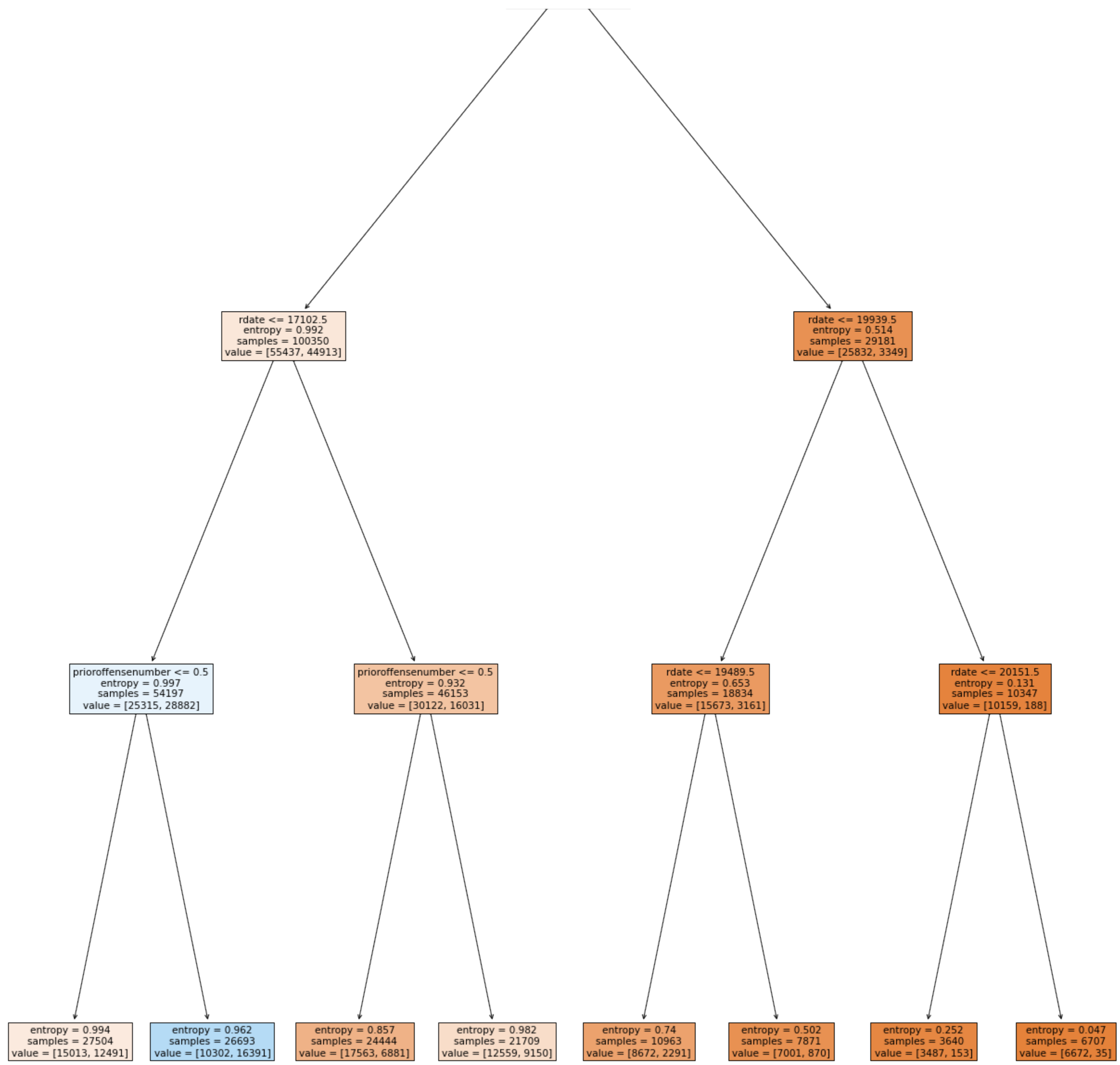
Out[26]:
```
▾                     DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_stat
e=0)
```

In [27]:
```
# Accuracy
print('Accuracy score:',accuracy_score(Y, clf.predict(X)))

# Residual value
resid_dev = np.sum(log_loss(Y, clf.predict_proba(X)))
print('Residual value:',resid_dev)
```

```
Accuracy score: 0.6744177069581799
Residual value: 0.5810712818689788
```

In [28]:
```
# Plot the tree
feature_names = X.columns
ax = subplots(figsize=(22,28))[1]
plot_tree(clf,
          feature_names=feature_names,
          ax=ax, filled = True);
```




rdate <= 17102.5
entropy = 0.992
samples = 100350
value = [55437, 44913]
rdate <= 19939.5
entropy = 0.514
samples = 29181
value = [25832, 3349]
prioroffensenumber <= 0.5
entropy = 0.997
samples = 54197
value = [25315, 28882]
prioroffensenumber <= 0.5
entropy = 0.932
samples = 46153
value = [30122, 16031]
rdate <= 19489.5
entropy = 0.653
samples = 18834
value = [15673, 3161]
rdate <= 20151.5
entropy = 0.131
samples = 10347
value = [10159, 188]
entropy = 0.994
samples = 27504
value = [15013, 12491]
entropy = 0.962
samples = 26693
value = [10302, 16391]
entropy = 0.857
samples = 24444
value = [17563, 6881]
entropy = 0.982
samples = 21709
value = [12559, 9150]
entropy = 0.74
samples = 10963
value = [8672, 2291]
entropy = 0.502
samples = 7871
value = [7001, 870]
entropy = 0.252
samples = 3640
value = [3487, 153]
entropy = 0.047
samples = 6707
value = [6672, 35]

In [29]:
```
# Convert feature_names (pandas Index) to a list
feature_names = feature_names.tolist()

# Print the decision tree rules
tree_rules = export_text(clf, feature_names=feature_names, show_weight
print(tree_rules)
```

```
|--- maxrdate <= 18907.50
|   |--- rdate <= 17102.50
|   |   |--- prioroffensenumber <= 0.50
|   |   |   |--- weights: [15013.00, 12491.00] class: 0.0
|   |   |--- prioroffensenumber >  0.50
|   |   |   |--- weights: [10302.00, 16391.00] class: 1.0
|   |--- rdate >  17102.50
|   |   |--- prioroffensenumber <= 0.50
|   |   |   |--- weights: [17563.00, 6881.00] class: 0.0
|   |   |--- prioroffensenumber >  0.50
|   |   |   |--- weights: [12559.00, 9150.00] class: 0.0
|--- maxrdate >  18907.50
|   |--- rdate <= 19939.50
|   |   |--- rdate <= 19489.50
|   |   |   |--- weights: [8672.00, 2291.00] class: 0.0
|   |   |--- rdate >  19489.50
|   |   |   |--- weights: [7001.00, 870.00] class: 0.0
|   |--- rdate >  19939.50
|   |   |--- rdate <= 20151.50
|   |   |   |--- weights: [3487.00, 153.00] class: 0.0
|   |   |--- rdate >  20151.50
|   |   |   |--- weights: [6672.00, 35.00] class: 0.0
```

In [30]:
```
# Pruning Classification Tree
# Load the packages
import sklearn.model_selection as skm
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplots
from sklearn.tree import (DecisionTreeClassifier as DTC,
                          DecisionTreeRegressor as DTR,
                          plot_tree,
                          export_text)
from sklearn.metrics import (accuracy_score,
                             log_loss)
from sklearn.ensemble import \
     (RandomForestRegressor as RF,
      GradientBoostingRegressor as GBR,
     RandomForestClassifier as RFC)
```

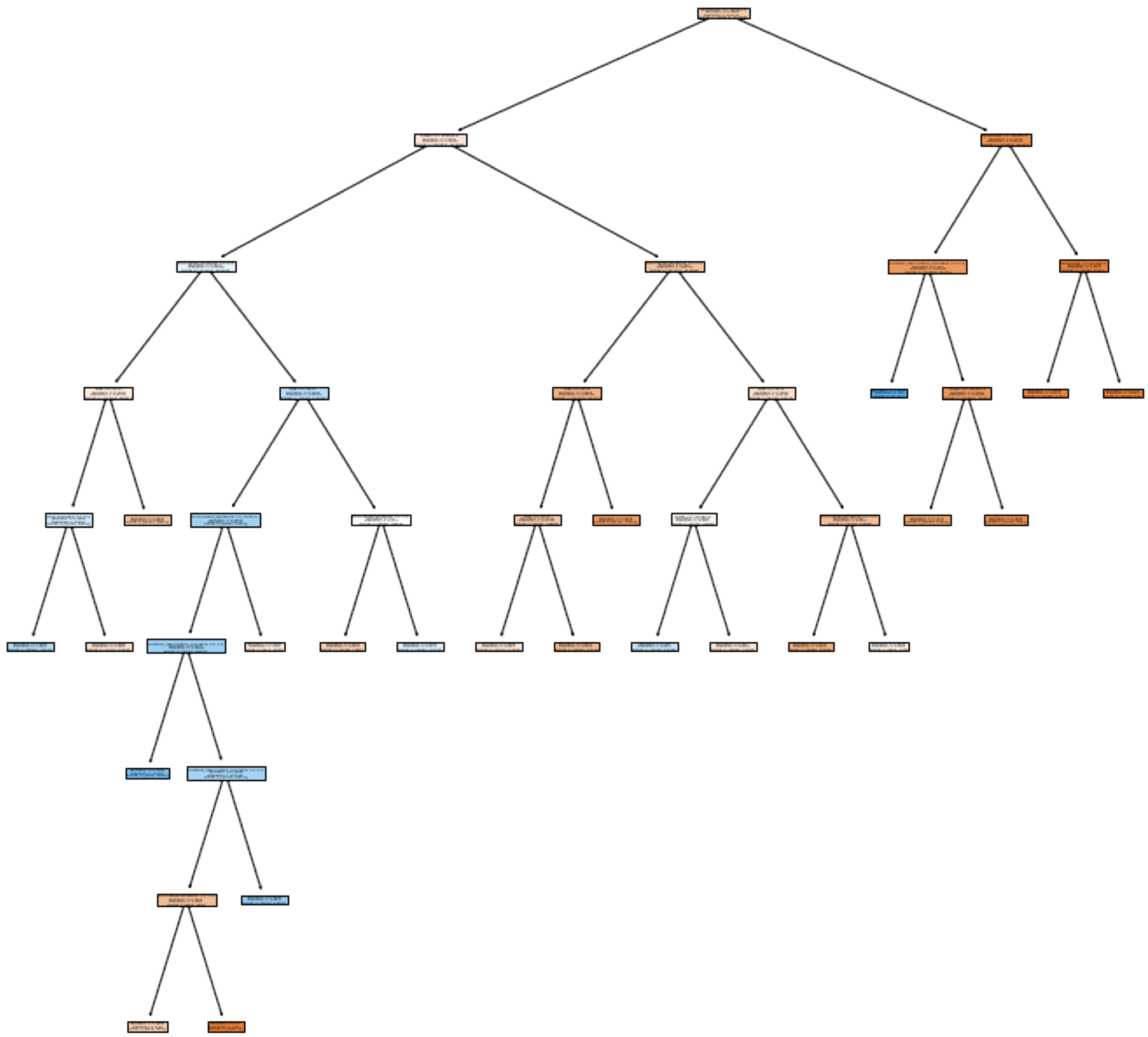
In [31]:
```
# Split the data between train and text
(X_train,
 X_test,
 Y_train,
 Y_test) = skm.train_test_split(X,
                                Y,
                                 test_size=0.3,
                                 random_state=0)

# Accuracy score
clf1 = DTC(criterion='entropy', random_state=0)
clf1.fit(X_train, Y_train)
accuracy_score(Y_test, clf.predict(X_test))
```

Out[31]: 0.674704065877509

In [32]:
```
ccp_path = clf.cost_complexity_pruning_path(X_train, Y_train)
kfold1 = skm.KFold(10,
                  random_state=1,
                  shuffle=True)

grid = skm.GridSearchCV(clf1,
                        {'ccp_alpha': ccp_path.ccp_alphas},
                        refit=True,
                        cv=kfold1,
                        scoring='accuracy')
grid.fit(X_train, Y_train)
grid.best_score_
```

Out[32]: 0.6982829875656346

In [33]:
```
# Plot the pruned tree
ax = subplots(figsize=(12, 12))[1]
best_ = grid.best_estimator_
plot_tree(best_,
          feature_names=feature_names,
          ax=ax, filled=True);
```



In [34]:
```
######################################## BAGGING AND RANDOM FOREST #####
```

In [35]:
```
# Building the forest
data_RF = RFC(max_features = X_train.shape[1], random_state=0)
data_RF.fit(X_train,Y_train)
```

Out[35]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_features=84, random_state=0)
```

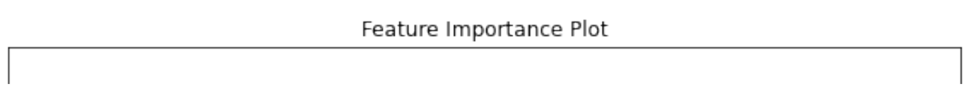
In [36]:
```
# Table importance
feature_imp = pd.DataFrame(
    {'importance':data_RF.feature_importances_},
    index=feature_names)
feature_imp.sort_values(by='importance', ascending=False)
```

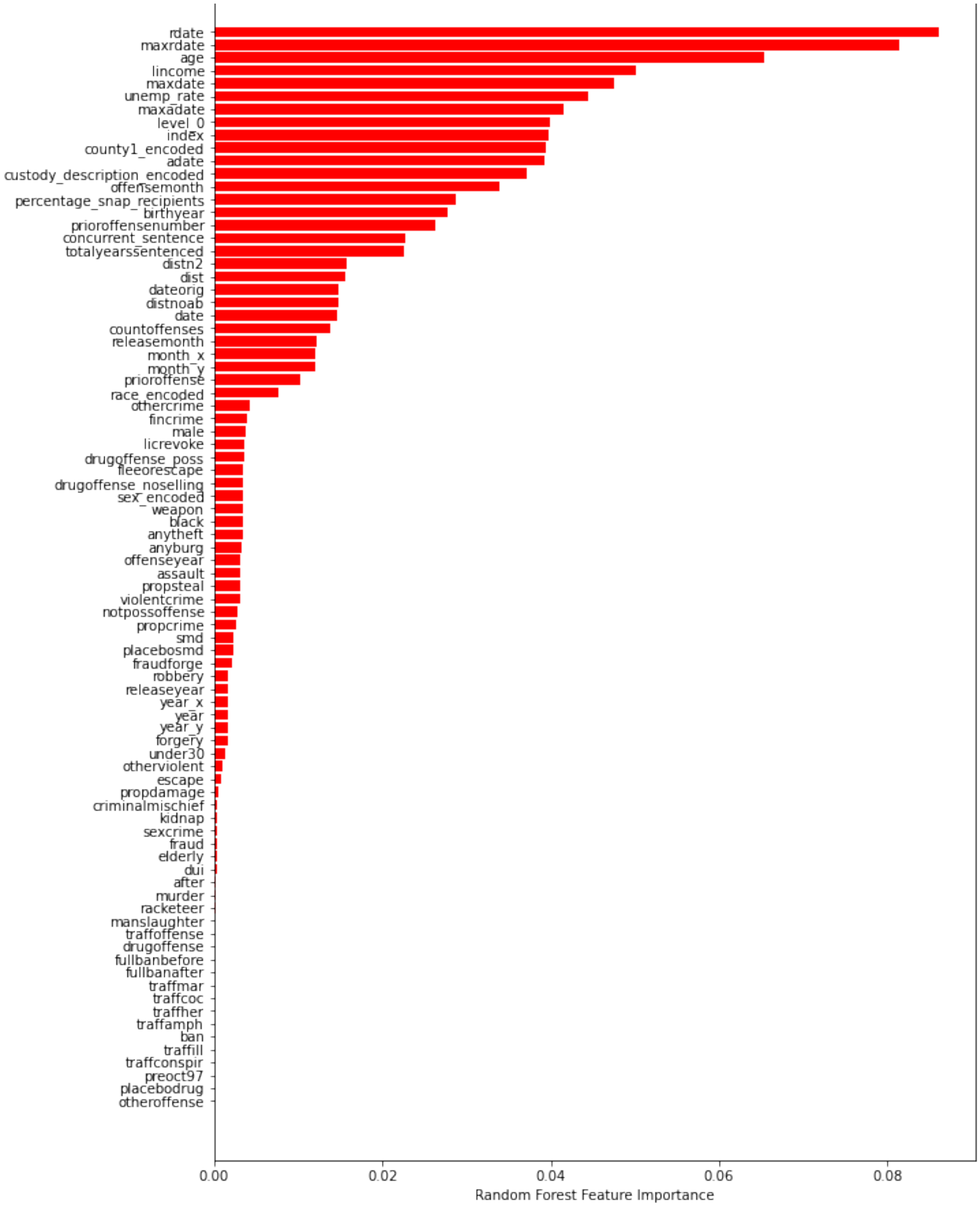
Out[36]:

	importance
rdate	0.086171
maxrdate	0.081360
age	0.065387
lincome	0.050155
maxdate	0.047501
...	...
traffher	0.000000
traffamph	0.000000
traffill	0.000000
traffconspir	0.000000
traffcoc	0.000000

84 rows × 1 columns

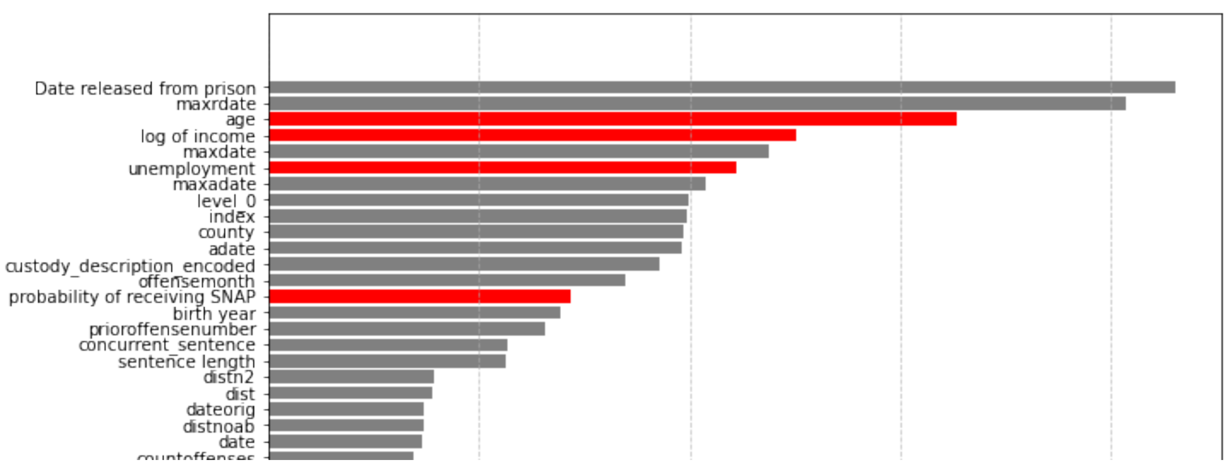
In [37]:
```
# Plot variable importance
importance = data_RF.feature_importances_
sorted_importance = importance.argsort()

plt.figure(figsize=(10, 16))
plt.barh(X.columns[sorted_importance], importance[sorted_importance],c
plt.xlabel('Random Forest Feature Importance')
plt.title('Feature Importance Plot')
plt.show()
```

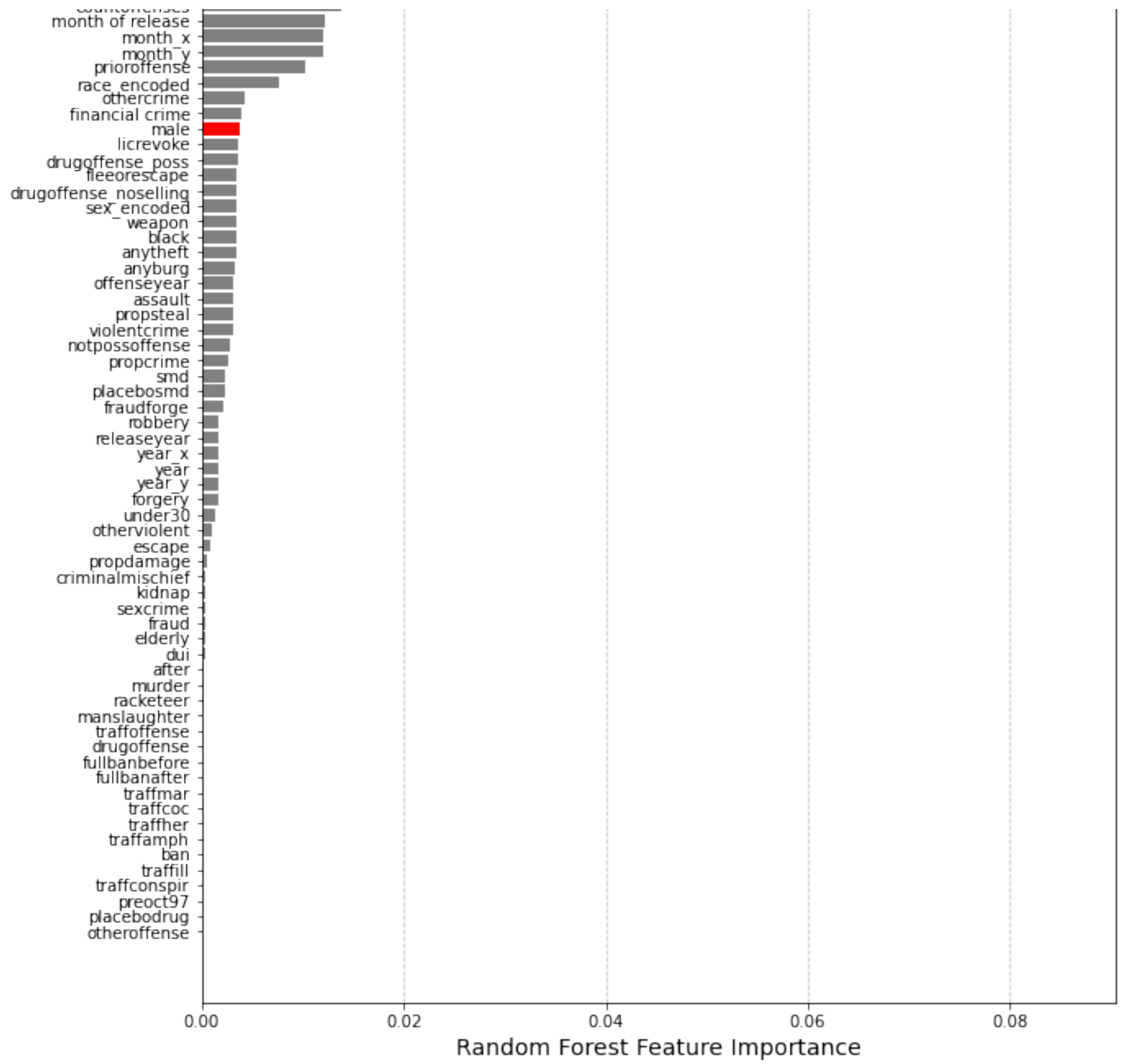




In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Assuming data_RF is your fitted Random Forest model and X is your fe
importance2 = data_RF.feature_importances_
sorted_importance2 = np.argsort(importance2)
```

```
# Create a DataFrame for better handling
feature_importance_df2 = pd.DataFrame({
    'Feature': X.columns[sorted_importance2],
    'Importance': importance[sorted_importance2]
})
# Dictionary for renaming multiple variables
rename_dict = {
    'unemp_rate': 'unemployment',
    'totalyearssentenced': 'sentence length',
    'fincrime': 'financial crime',
    'priooffensenumber':'number of prior offenses',
    'birthyear': 'birth year',
    'county1_encoded':'county',
    'releasemonth': 'month of release',
    'rdate':'Date released from prison',
    'lincome':'log of income',
    'percentage_snap_recipients':'probability of receiving SNAP'
}
# Change 'unemp_rate' to 'employment'
feature_importance_df2['Feature'] = feature_importance_df2['Feature'].

# Highlighted features
highlighted_features = ['age', 'unemployment','male', 'log of income',

# Create a color array: Use a bright color for highlighted features an
colors = ['red' if feature in highlighted_features else 'grey' for fea

plt.figure(figsize=(10, 16))
plt.barh(feature_importance_df2['Feature'], feature_importance_df2['Im
plt.xlabel('Random Forest Feature Importance', fontsize=14)
#plt.title('Feature Importance Plot', fontsize=16)
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Show plot
plt.savefig('importance.png', format='png', dpi=300, bbox_inches='tigh
plt.show();
```





In [39]:
```
############################################ SUMMARY STATISTICS ######
```

In [40]:
```
# Adjusting column names to match the dataset and re-running the analy
columns_of_interest = [ 'income', 'percentage_snap_recipients', 'anyre

# Filtering the dataset for the relevant columns and grouping by 'anyr
summary_stats = data[columns_of_interest].groupby('anyrecid').describe

# Display the summary statistics
print(summary_stats)
```

```
           income                                                   \
            count          mean          std      min      25%      5
0%
anyrecid
0.0       81269.0  43970.506048  5890.209282  24031.0  40589.0  4418
6.0
1.0       48262.0  42110.100141  5792.292067  23852.0  38457.0  4231
1.0

                                percentage_snap_recipients           \
              75%      max                          count      mean
std
anyrecid
0.0       47876.0  67967.0                         81269.0  0.262735  0.11
8334
1.0       45995.0  61379.0                         48262.0  0.185716  0.08
9515


               min       25%       50%       75%       max
anyrecid
0.0       0.109079  0.142761  0.246107  0.393976  0.417409
1.0       0.109079  0.131741  0.147436  0.187432  0.417409
```

In [41]:
```
########################################### PAPER REGRESSIONS #######
```

In [42]:
```
# Load needed packages
import statsmodels.formula.api as smf
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
# Regression paper
reg = smf.ols('Y ~ after + dist + after * dist + unemp_rate + after *
print(reg.summary())
```

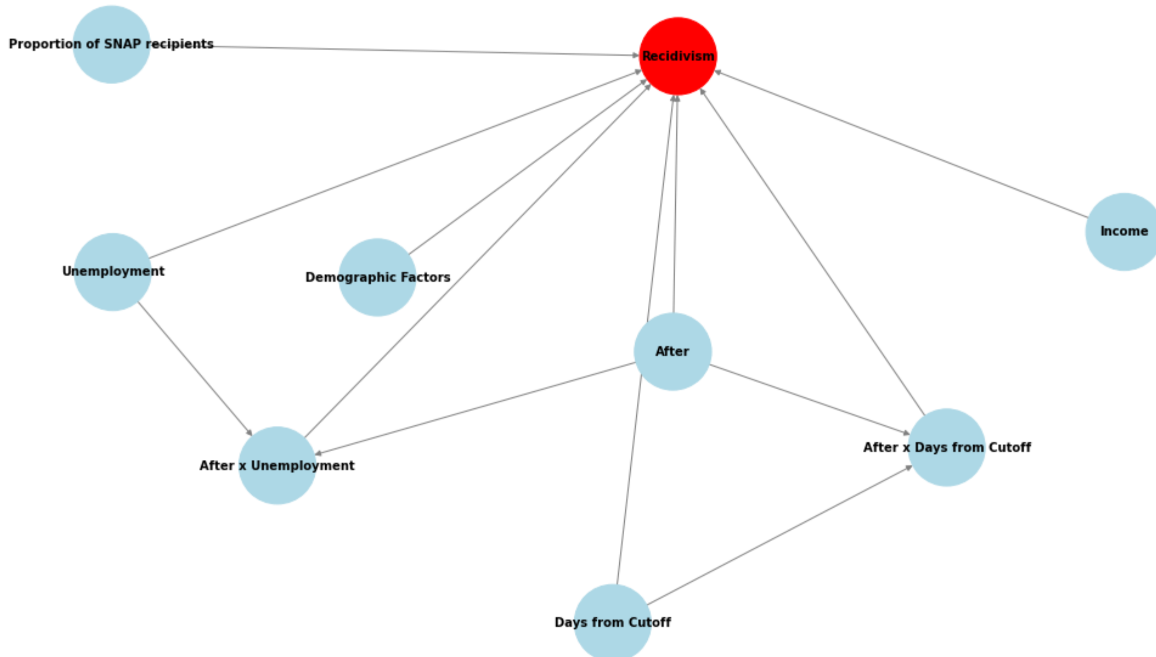
```
                            OLS Regression Results
======================================================================
=========
```

```
Dep. Variable:                      Y   R-squared:
0.114
Model:                            OLS   Adj. R-squared:
0.114
Method:                 Least Squares   F-statistic:
2373.
Date:                Fri, 20 Dec 2024   Prob (F-statistic):
0.00
Time:                        20:13:10   Log-Likelihood:
-81850.
No. Observations:              129531   AIC:
1.637e+05
Df Residuals:                  129523   BIC:
1.638e+05
Df Model:                           7
Covariance Type:            nonrobust
======================================================================
=========================
                                 coef    std err          t      P>|t
|      [0.025      0.975]
----------------------------------------------------------------------
-------------------------
Intercept                      1.6524      0.111     14.874      0.00
0       1.435       1.870
after                         -0.0944      0.025     -3.780      0.00
0      -0.143      -0.045
dist                       -4.374e-05   7.71e-05     -0.567      0.57
1      -0.000       0.000
after:dist                  2.109e-05   7.72e-05      0.273      0.78
5      -0.000       0.000
unemp_rate                    -0.0085      0.005     -1.718      0.08
6      -0.018       0.001
after:unemp_rate               0.0226      0.005      4.540      0.00
0       0.013       0.032
lincome                       -0.0841      0.010     -8.203      0.00
0      -0.104      -0.064
percentage_snap_recipients    -1.3006      0.024    -53.121      0.00
0      -1.349      -1.253
======================================================================
=========
Omnibus:                   977370.264   Durbin-Watson:
1.875
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
13963.171
Skew:                           0.363   Prob(JB):
0.00
Kurtosis:                       1.565   Cond. No.
4.27e+05
======================================================================
=========
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors i
s correctly specified.
[2] The condition number is large, 4.27e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

In [44]:
```
################################# DIRECTED ACYCLIC GRAPH ############
```

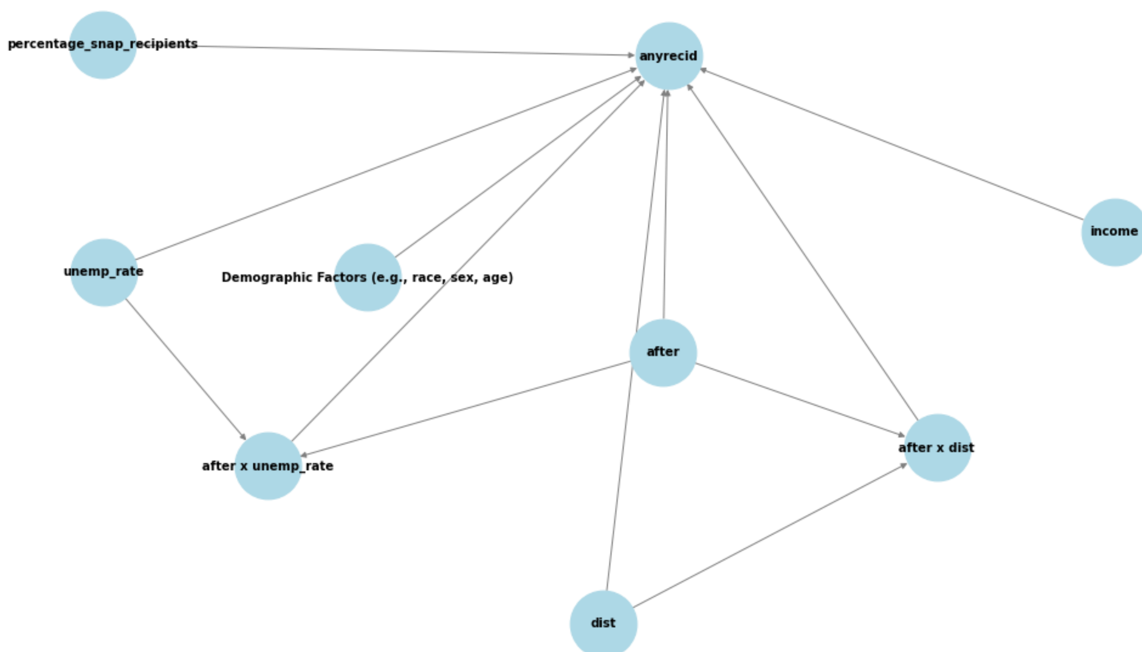
In [49]:
```
# Importing necessary libraries
import matplotlib.pyplot as plt
import networkx as nx

# Create a directed acyclic graph (DAG)
G = nx.DiGraph()

# Adding nodes for variables
nodes = [
    "Recidivism", "After", "Days from Cutoff", "Unemployment", "Income
    "Proportion of SNAP recipients", "After x Days from Cutoff", "Afte
    "Demographic Factors"
]
G.add_nodes_from(nodes)

# Adding edges to represent relationships based on the equation
edges = [
    ("After", "Recidivism"),
    ("Days from Cutoff", "Recidivism"),
    ("Unemployment", "Recidivism"),
    ("Income", "Recidivism"),
    ("Proportion of SNAP recipients", "Recidivism"),
    ("After", "After x Days from Cutoff"),
    ("Days from Cutoff", "After x Days from Cutoff"),
    ("After x Days from Cutoff", "Recidivism"),
    ("After", "After x Unemployment"),
    ("Unemployment", "After x Unemployment"),
    ("After x Unemployment", "Recidivism"),
    ("Demographic Factors", "Recidivism")
]
G.add_edges_from(edges)

# Plotting the DAG
plt.figure(figsize=(14, 8))
pos = nx.spring_layout(G, seed=42)
node_colors = ["red" if node == "Recidivism" else "lightblue" for node
nx.draw(G, pos, with_labels=True, node_size=4000, node_color=node_colo
plt.savefig('Directed_DAG.png',format ='png',dpi = 300,bbox_inches='ti
plt.show()
```



In [50]:
```
# Importing necessary libraries
import matplotlib.pyplot as plt
import networkx as nx

# Create a directed acyclic graph (DAG)
G1 = nx.DiGraph()

# Adding nodes for variables based on dataset column names
nodes = [
    "anyrecid", "after", "dist", "unemp_rate", "income",
    "percentage_snap_recipients", "after x dist", "after x unemp_rate"
    "Demographic Factors (e.g., race, sex, age)"
]
G1.add_nodes_from(nodes)

# Adding edges to represent relationships based on the equation
edges = [
    ("after", "anyrecid"),
    ("dist", "anyrecid"),
    ("unemp_rate", "anyrecid"),
    ("income", "anyrecid"),
    ("percentage_snap_recipients", "anyrecid"),
    ("after", "after x dist"),
    ("dist", "after x dist"),
    ("after x dist", "anyrecid"),
    ("after", "after x unemp_rate"),
    ("unemp_rate", "after x unemp_rate"),
```

```
    ("after x unemp_rate", "anyrecid"),
    ("Demographic Factors (e.g., race, sex, age)", "anyrecid")
]
G1.add_edges_from(edges)

# Plotting the DAG
plt.figure(figsize=(14, 8))
pos = nx.spring_layout(G1, seed=42)
nx.draw(G1, pos, with_labels=True, node_size=3000, node_color="lightbl
plt.title("Directed Acyclic Graph (DAG) for Recidivism Model (Updated
plt.show()
```

Directed Acyclic Graph (DAG) for Recidivism Model (Updated with Dataset Variables)



In [51]:
```
############################################ BACKDOOR #################
```

In [52]:
```
# Load packages
from dowhy import CausalModel
```

In [53]:
```
# Setting causal model
model = CausalModel(data=data,
                    treatment='after',
                    outcome='anyrecid',
                    common_causes=['unemp_rate','lincome','percentage_
                    graph=G1)

# Identify the estimand
estimand = model.identify_effect()
```

```
/Users/edithsimochemo/opt/anaconda3/lib/python3.9/site-packages/dowh
y/causal_model.py:582: UserWarning: 3 variables are assumed unobserve
d because they are not in the dataset. Configure the logging level to
`logging.WARNING` or higher for additional details.
  warnings.warn(
```

In [54]:
```
import warnings

# Display the identified estimand
print(estimand)

# Suppress all warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    # Obtain estimates
    estimate = model.estimate_effect(identified_estimand=estimand,
                                     method_name='backdoor.linear_regr
    print(estimate)
```

```
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
   d
────────(E[anyrecid])
d[after]
Estimand assumption 1, Unconfoundedness: If U→{after} and U→anyrecid
then P(anyrecid|after,,U) = P(anyrecid|after,)

### Estimand : 2
Estimand name: iv
No such variable(s) found!

### Estimand : 3
Estimand name: frontdoor
No such variable(s) found!
```

```
*** Causal Estimate ***

## Identified estimand
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
   d
────────(E[anyrecid])
d[after]
Estimand assumption 1, Unconfoundedness: If U→{after} and U→anyrecid
then P(anyrecid|after,,U) = P(anyrecid|after,)

## Realized estimand
b: anyrecid~after+after*unemp_rate+after*dist+after*percentage_snap_r
ecipients+after*income
Target units:

## Estimate
Mean value: -0.18008907485851255
### Conditional Estimates
__categorical__unemp_rate  __categorical__dist  __categorical__percen
tage_snap_recipients  __categorical__income
(1.399, 3.3]              (-0.001, 1205.0]     (0.108, 0.135]
(23851.999, 38477.0]    -0.003388

(38477.0, 41960.0]      -0.008728

(41960.0, 44850.0]      -0.011889

(44850.0, 48478.0]      -0.023568

(48478.0, 67967.0]      -0.026729

...
(8.9, 22.6]               (4576.0, 6857.0]     (0.389, 0.417]
(23851.999, 38477.0]    -0.366911

(38477.0, 41960.0]      -0.386116

(41960.0, 44850.0]      -0.390642

(44850.0, 48478.0]      -0.396940

(48478.0, 67967.0]      -0.410806
Length: 391, dtype: float64
```

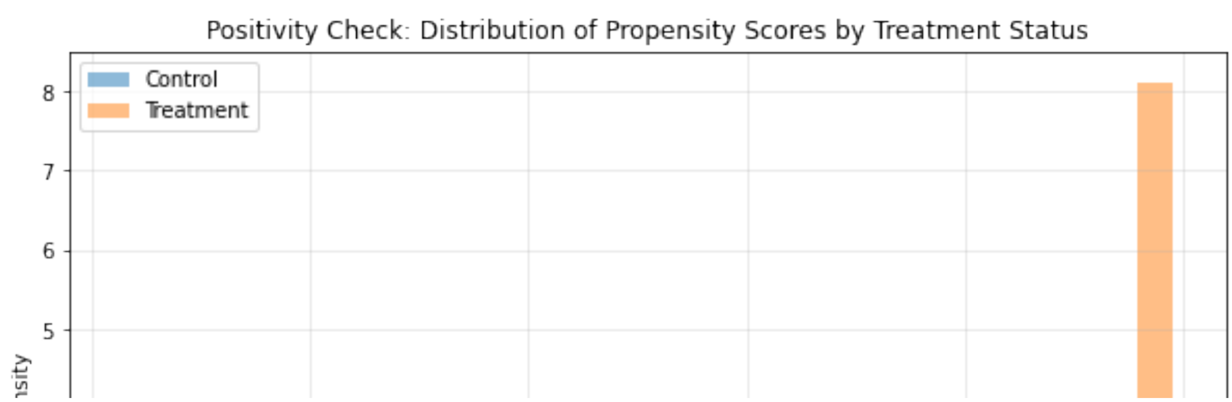
In [55]:
```
################################## INVERSE PROBABILITY WEIGHTING ######
```

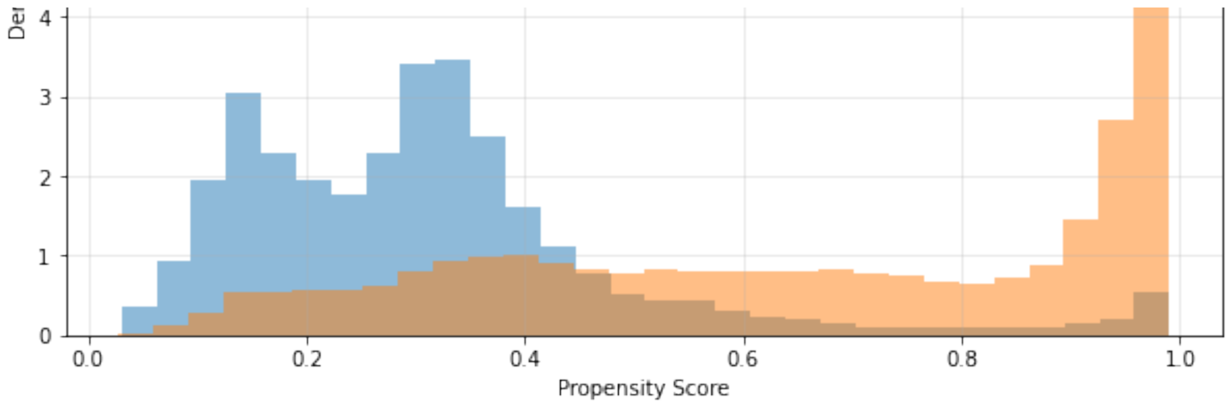
In [69]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

df = pd.read_stata('final_fl2.dta')

def calculate_propensity_scores(df, treatment, covariates, outcome):
    """
    Calculate propensity scores and weights for causal inference analy

    Parameters:
    -----------
    df : pandas.DataFrame
        Input dataset
    treatment : str
        Name of treatment variable column
    covariates : list
        List of covariate column names
    outcome : str
        Name of outcome variable column

    Returns:
    --------
    tuple
        (DataFrame with weights, propensity scores array, ATE estimate
    """
    # Create a copy to avoid modifying original data
    df_clean = df.copy()

    # Drop rows with missing values
    df_clean = df_clean.dropna(subset=[treatment] + covariates + [outc

    # Standardize covariates
    scaler = StandardScaler()
    X = scaler.fit_transform(df_clean[covariates])
    X = pd.DataFrame(X, columns=covariates, index=df_clean.index)
    y = df_clean[treatment]

    # Fit logistic regression with balanced class weights
    logit = LogisticRegression(class_weight='balanced', random_state=4
    logit.fit(X, y)

    # Calculate propensity scores
    propensity_scores = logit.predict_proba(X)[:, 1]

    # Trim extreme propensity scores to avoid infinite weights
```

```
    eps = 0.01
    propensity_scores = np.clip(propensity_scores, eps, 1 - eps)

    # Calculate inverse probability weights
    weights = np.where(y == 1,
                       1/propensity_scores,
                       1/(1 - propensity_scores))

    # Add scores and weights to dataframe
    df_clean['propensity_score'] = propensity_scores
    df_clean['weights'] = weights

    # Calculate ATE
    ate = np.average(df_clean[outcome], weights=df_clean['weights'])

    return df_clean, propensity_scores, ate

def plot_positivity_check(df, treatment_col='after'):
    """
    Create positivity check plot for propensity scores.

    Parameters:
    -----------
    df : pandas.DataFrame
        DataFrame containing propensity scores
    treatment_col : str
        Name of treatment variable column
    """
    plt.figure(figsize=(10, 6))

    # Create separate density plots for each group
    for group in [0, 1]:
        mask = df[treatment_col] == group
        plt.hist(df.loc[mask, 'propensity_score'],
                bins=30,
                density=True,
                alpha=0.5,
                label=f"{'Treatment' if group == 1 else 'Control'}")

    plt.title('Positivity Check: Distribution of Propensity Scores by
    plt.xlabel('Propensity Score')
    plt.ylabel('Density')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.savefig('positivity_check.png', format = 'png', dpi =300, bbox
    plt.show()


if __name__ == "__main__":
    # Define variables
```

```
    treatment = 'after'
    covariates = ['age', 'male', 'black',
                  'prioroffense', 'unemp_rate','lincome','percentage_
    outcome = 'anyrecid'

    # Calculate propensity scores and weights
    df_weighted, prop_scores, ate = calculate_propensity_scores(
        df, treatment, covariates, outcome
    )

    # Display results
    print("\nSample of propensity scores and weights:")
    print(df_weighted[['offenderid', treatment, 'propensity_score', 'w

    print(f"\nEstimated Average Treatment Effect (ATE): {ate:.4f}")

    # Create positivity check plot
    plot_positivity_check(df_weighted, treatment)

    # Print covariate balance summary
    print("\nCovariate balance summary:")
    for cov in covariates:
        treated_mean = np.average(df_weighted.loc[df_weighted[treatmen
                                weights=df_weighted.loc[df_weighted[tr
        control_mean = np.average(df_weighted.loc[df_weighted[treatmen
                                weights=df_weighted.loc[df_weighted[tr
        print(f"{cov}: Treated mean = {treated_mean:.2f}, Control mear
```

```
Sample of propensity scores and weights:
  offenderid  after  propensity_score   weights
1    A000043    1.0          0.137165  7.290506
2    A000043    1.0          0.938853  1.065130
3    A000077    1.0          0.368211  2.715831
5    A000093    0.0          0.393505  1.648819
6    A000101    1.0          0.516569  1.935851

Estimated Average Treatment Effect (ATE): 0.3769
```






```
Covariate balance summary:
age: Treated mean = 32.26, Control mean = 33.30
male: Treated mean = 0.90, Control mean = 0.92
black: Treated mean = 0.52, Control mean = 0.50
prioroffense: Treated mean = 0.38, Control mean = 0.37
unemp_rate: Treated mean = 5.27, Control mean = 5.68
lincome: Treated mean = 10.59, Control mean = 10.64
percentage_snap_recipients: Treated mean = 0.20, Control mean = 0.24
```

In [63]:
```
################################################# METALEARNERS ########
```

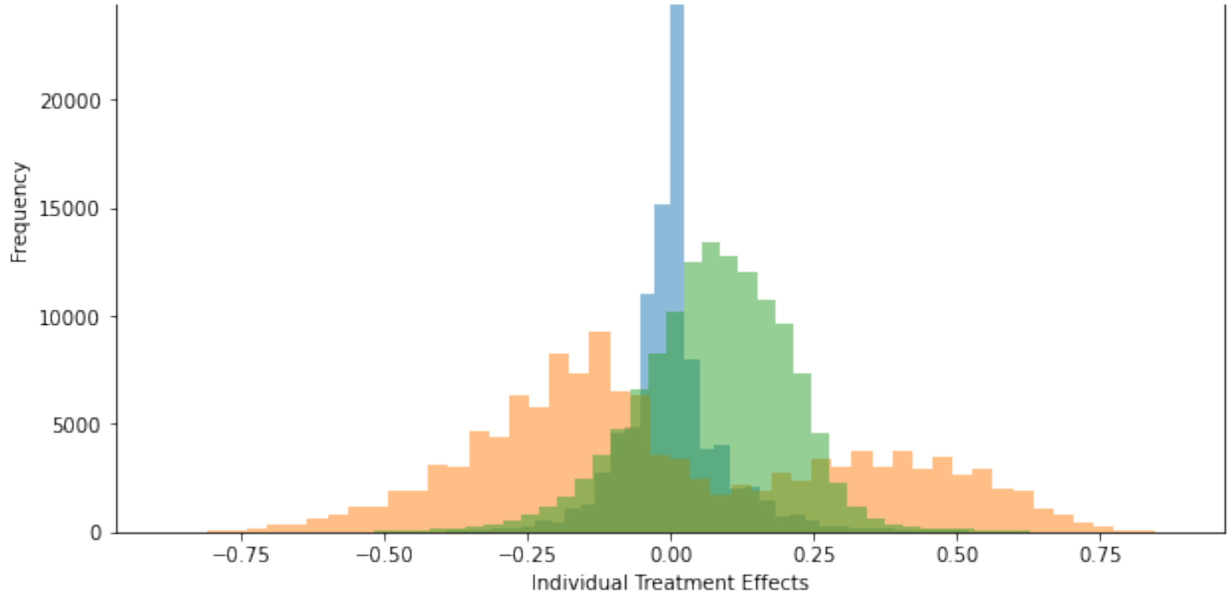
In [74]:
```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

df = pd.read_stata('final_fl2.dta')

# Modify features based on available columns
features = ['concurrent_sentence', 'drugoffense', 'traffoffense', 'oth
            'age', 'male', 'black', 'totalyearssentenced', 'prioroffens
            'lincome','percentage_snap_recipients']
treatment = 'after'
outcome = 'anyrecid'

# Drop rows with missing values
df_clean = data[features + [treatment, outcome]].dropna()

# Split features
X = df_clean[features]
T = df_clean[treatment]
Y = df_clean[outcome]

# S-Learner
```

```
def s_learner(X, T, Y):
    # Combine features with treatment
    X_s = X.copy()
    X_s['treatment'] = T

    # Train random forest
    s_model = RandomForestRegressor(n_estimators=100, random_state=42)
    s_model.fit(X_s, Y)

    # Predict potential outcomes
    X_s_1 = X_s.copy()
    X_s_0 = X_s.copy()
    X_s_1['treatment'] = 1
    X_s_0['treatment'] = 0

    y1_pred = s_model.predict(X_s_1)
    y0_pred = s_model.predict(X_s_0)

    # Calculate ATE
    ate = np.mean(y1_pred - y0_pred)
    return ate, y1_pred, y0_pred

# T-Learner
def t_learner(X, T, Y):
    # Split data by treatment
    X_t = X[T == 1]
    X_c = X[T == 0]
    Y_t = Y[T == 1]
    Y_c = Y[T == 0]

    # Train separate models
    t1_model = RandomForestRegressor(n_estimators=100, random_state=42
    t0_model = RandomForestRegressor(n_estimators=100, random_state=42

    t1_model.fit(X_t, Y_t)
    t0_model.fit(X_c, Y_c)

    # Predict potential outcomes
    y1_pred = t1_model.predict(X)
    y0_pred = t0_model.predict(X)

    # Calculate ATE
    ate = np.mean(y1_pred - y0_pred)
    return ate, y1_pred, y0_pred

# X-Learner
def x_learner(X, T, Y):
    # First stage: T-Learner
    t_ate, y1_pred, y0_pred = t_learner(X, T, Y)
```

```
    # Second stage: Calculate individual treatment effects
    X_t = X[T == 1]
    X_c = X[T == 0]
    Y_t = Y[T == 1]
    Y_c = Y[T == 0]

    # Calculate residuals
    D1 = Y_t - y0_pred[T == 1]
    D0 = y1_pred[T == 0] - Y_c

    # Train second stage models
    x1_model = RandomForestRegressor(n_estimators=100, random_state=42
    x0_model = RandomForestRegressor(n_estimators=100, random_state=42

    x1_model.fit(X_t, D1)
    x0_model.fit(X_c, D0)

    # Predict treatment effects
    tau1 = x1_model.predict(X)
    tau0 = x0_model.predict(X)

    # Calculate final treatment effect
    g = np.mean(T)  # Propensity score (simplified)
    tau = g * tau0 + (1 - g) * tau1

    return np.mean(tau), tau

# Run all models
print("Running models...")
s_ate, s_y1, s_y0 = s_learner(X, T, Y)
t_ate, t_y1, t_y0 = t_learner(X, T, Y)
x_ate, x_tau = x_learner(X, T, Y)

# Create results summary
results = pd.DataFrame({
    'Model': ['S-Learner', 'T-Learner', 'X-Learner'],
    'Average Treatment Effect': [s_ate, t_ate, np.mean(x_tau)]
})

print("\
Model Results:")
print(results)

# Visualize treatment effects distribution
plt.figure(figsize=(10, 6))
plt.hist(s_y1 - s_y0, bins=50, alpha=0.5, label='S-Learner')
plt.hist(t_y1 - t_y0, bins=50, alpha=0.5, label='T-Learner')
plt.hist(x_tau, bins=50, alpha=0.5, label='X-Learner')
plt.xlabel('Individual Treatment Effects')
plt.ylabel('Frequency')
```

```
plt.legend()
plt.ylim(0, 30000)
plt.savefig('metalearners.png', format = 'png', dpi =300, bbox_inches=
plt.show()


# Calculate model performance metrics
def calculate_metrics(y_true, y_pred_treated, y_pred_control):
    mse_treated = mean_squared_error(y_true[T == 1], y_pred_treated[T
    mse_control = mean_squared_error(y_true[T == 0], y_pred_control[T
    return np.sqrt(mse_treated), np.sqrt(mse_control)

s_rmse_t, s_rmse_c = calculate_metrics(Y, s_y1, s_y0)
t_rmse_t, t_rmse_c = calculate_metrics(Y, t_y1, t_y0)

metrics = pd.DataFrame({
    'Model': ['S-Learner', 'T-Learner'],
    'RMSE (Treated)': [s_rmse_t, t_rmse_t],
    'RMSE (Control)': [s_rmse_c, t_rmse_c]
})

print("\
Model Performance Metrics:")
print(metrics)

# Feature importance for S-Learner
s_model = RandomForestRegressor(n_estimators=100, random_state=42)
X_s = X.copy()
X_s['treatment'] = T
s_model.fit(X_s, Y)

feature_importance = pd.DataFrame({
    'Feature': features + ['treatment'],
    'Importance': s_model.feature_importances_
}).sort_values('Importance', ascending=False)

print("\
Feature Importance (S-Learner):")
print(feature_importance)
```

```
Running models...
Model Results:
       Model  Average Treatment Effect
0  S-Learner                  0.002248
1  T-Learner                  0.000301
2  X-Learner                  0.076148
```




```
Model Performance Metrics:
       Model  RMSE (Treated)  RMSE (Control)
0  S-Learner        0.173536        0.186604
1  T-Learner        0.173503        0.189568
Feature Importance (S-Learner):
                      Feature  Importance
11  percentage_snap_recipients    0.258495
4                          age    0.161238
10                     lincome    0.160923
9                   unemp_rate    0.156308
0           concurrent_sentence    0.101182
7           totalyearssentenced    0.100694
6                        black    0.021428
8                 prioroffense    0.021363
5                         male    0.013172
12                   treatment    0.005197
1                 drugoffense    0.000000
2                 traffoffense    0.000000
3                 otheroffense    0.000000
```

In [73]:

```
############################################## BOOTSTRAP ############
```

In [79]:

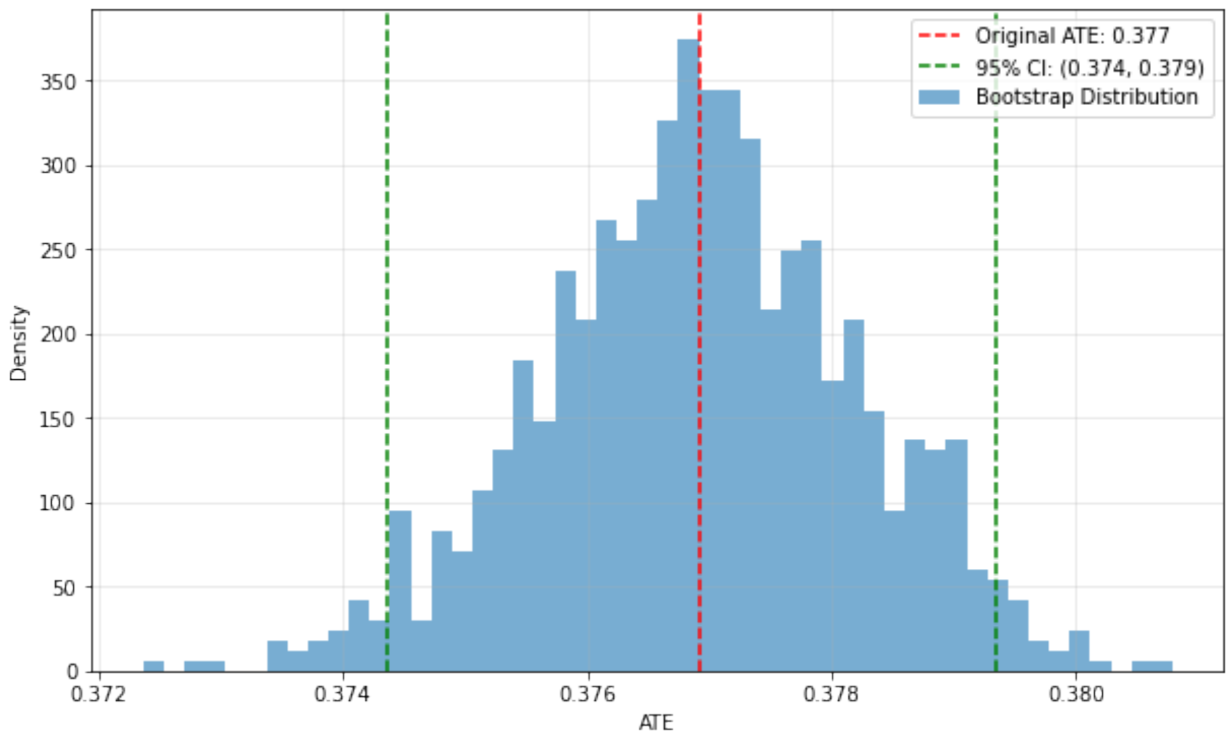
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from typing import Tuple, List, Optional
import warnings

def calculate_propensity_scores(df: pd.DataFrame,
```

```
                              treatment: str,
                              covariates: List[str],
                              outcome: str) -> Tuple[pd.DataFrame, np.
    """
    Calculate propensity scores and weights for causal inference analy

    Parameters and returns same as before
    """
    # Previous implementation remains the same
    df_clean = df.copy()
    df_clean = df_clean.dropna(subset=[treatment] + covariates + [outc

    scaler = StandardScaler()
    X = scaler.fit_transform(df_clean[covariates])
    X = pd.DataFrame(X, columns=covariates, index=df_clean.index)
    y = df_clean[treatment]

    logit = LogisticRegression(class_weight='balanced', random_state=4
    logit.fit(X, y)

    propensity_scores = logit.predict_proba(X)[:, 1]
    eps = 0.01
    propensity_scores = np.clip(propensity_scores, eps, 1 - eps)

    weights = np.where(y == 1,
                      1/propensity_scores,
                      1/(1 - propensity_scores))

    df_clean['propensity_score'] = propensity_scores
    df_clean['weights'] = weights

    ate = np.average(df_clean[outcome], weights=weights)

    return df_clean, propensity_scores, ate

def bootstrap_ate(df: pd.DataFrame,
                 treatment: str,
                 outcome: str,
                 covariates: List[str],
                 n_bootstrap: int = 1000,
                 random_state: Optional[int] = None) -> Tuple[float, f
    """
    Perform bootstrap analysis of Average Treatment Effect (ATE).

    Parameters:
    -----------
    df : pandas.DataFrame
        Input dataset
    treatment : str
        Name of treatment variable column
```

```
    outcome : str
        Name of outcome variable column
    covariates : List[str]
        List of covariate column names
    n_bootstrap : int, optional
        Number of bootstrap iterations (default: 1000)
    random_state : int, optional
        Random seed for reproducibility

    Returns:
    --------
    Tuple[float, float, float, np.ndarray]
        (original ATE, lower CI, upper CI, bootstrap ATEs)
    """
    if random_state is not None:
        np.random.seed(random_state)

    # Calculate original ATE
    df_weighted, _, original_ate = calculate_propensity_scores(
        df, treatment, covariates, outcome
    )

    # Perform bootstrap
    bootstrap_ates = []
    for _ in range(n_bootstrap):
        # Sample with replacement
        bootstrap_indices = np.random.choice(
            len(df_weighted),
            size=len(df_weighted),
            replace=True
        )
        bootstrap_sample = df_weighted.iloc[bootstrap_indices]

        # Recalculate propensity scores and weights for bootstrap samp
        _, _, bootstrap_ate = calculate_propensity_scores(
            bootstrap_sample,
            treatment,
            covariates,
            outcome
        )
        bootstrap_ates.append(bootstrap_ate)

    # Calculate confidence intervals
    ci_lower, ci_upper = np.percentile(bootstrap_ates, [2.5, 97.5])

    return original_ate, ci_lower, ci_upper, np.array(bootstrap_ates)

def plot_bootstrap_results(original_ate: float,
                           bootstrap_ates: np.ndarray,
                           ci_lower: float,
```

```
                          ci_upper: float) -> None:
    """
    Plot bootstrap distribution with confidence intervals.

    Parameters:
    -----------
    original_ate : float
        Original ATE estimate
    bootstrap_ates : numpy.ndarray
        Array of bootstrap ATE estimates
    ci_lower : float
        Lower bound of confidence interval
    ci_upper : float
        Upper bound of confidence interval
    """
    plt.figure(figsize=(10, 6))

    # Plot histogram of bootstrap estimates
    plt.hist(bootstrap_ates, bins=50, density=True, alpha=0.6,
             label='Bootstrap Distribution')

    # Add vertical lines for original ATE and CIs
    plt.axvline(original_ate, color='red', linestyle='dashed',
                label=f'Original ATE: {original_ate:.3f}')
    plt.axvline(ci_lower, color='green', linestyle='dashed',
                label=f'95% CI: ({ci_lower:.3f}, {ci_upper:.3f})')
    plt.axvline(ci_upper, color='green', linestyle='dashed')


    plt.xlabel('ATE')
    plt.ylabel('Density')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.savefig('bootstrap_int.png',format ='png', dpi = 300,bbox_inch
    plt.show()

# Example usage:
if __name__ == "__main__":
    # Define variables
    treatment = 'after'
    covariates = ['age', 'male', 'black',
                  'prioroffense', 'unemp_rate','lincome','percentage_
    outcome = 'anyrecid'

    # Perform bootstrap analysis
    original_ate, ci_lower, ci_upper, bootstrap_ates = bootstrap_ate(
        df,
        treatment,
        outcome,
        covariates,
```

```
        n_bootstrap=1000,
        random_state=42
    )

    # Plot results
    plot_bootstrap_results(original_ate, bootstrap_ates, ci_lower, ci_

    # Print numerical results
    print(f"Original ATE: {original_ate:.3f}")
    print(f"95% Confidence Interval: ({ci_lower:.3f}, {ci_upper:.3f})"
    print(f"Standard Error: {np.std(bootstrap_ates):.3f}")

    # Additional statistics
    print(f"\nBootstrap Statistics:")
    print(f"Mean of bootstrap estimates: {np.mean(bootstrap_ates):.3f}
    print(f"Median of bootstrap estimates: {np.median(bootstrap_ates):
    print(f"Standard deviation of bootstrap estimates: {np.std(bootstr
```




```
Original ATE: 0.377
95% Confidence Interval: (0.374, 0.379)
Standard Error: 0.001

Bootstrap Statistics:
Mean of bootstrap estimates: 0.377
Median of bootstrap estimates: 0.377
Standard deviation of bootstrap estimates: 0.001
```

In [76]:
```
############################################ DOUBLE ML MODELS #########
```

In [85]:
```
# librairies
from econml.dml import DML
from sklearn.linear_model import RidgeCV, LinearRegression

# Set up features and clean data
features = ['age', 'black', 'male', 'prioroffense', 'totalyearssentenc
            'prioroffensenumber', 'countoffenses','unemp_rate','lincome
df_clean = df.dropna(subset=['after', 'anyrecid'] + features)

# Prepare variables
X = df_clean[features]
T = df_clean['after'].astype(float)
Y = df_clean['anyrecid'].astype(float)

print("Data shape after cleaning:", X.shape)
print("\
Treatment variable (after) statistics:")
print(T.describe())
print("\
Outcome variable statistics:")
print(Y.describe())

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Linear Double ML Model
dml_linear = DML(
    model_y=LassoCV(random_state=123, max_iter=2000),
    model_t=LassoCV(random_state=123, max_iter=2000),
    model_final=LinearRegression(fit_intercept=True),
    random_state=123
)

# Fit and get linear model results
dml_linear.fit(Y, T, X=X_scaled)
effect_linear = dml_linear.effect(X=X_scaled)

print("\
Linear Double ML Results:")
print("Average Treatment Effect:", np.mean(effect_linear))

# Non-Linear Double ML Model
dml_nonlinear = DML(
    model_y=RandomForestRegressor(n_estimators=200, max_depth=5, rando
    model_t=RandomForestRegressor(n_estimators=200, max_depth=5, rando
    model_final=LinearRegression(fit_intercept=True),
    random_state=123
```

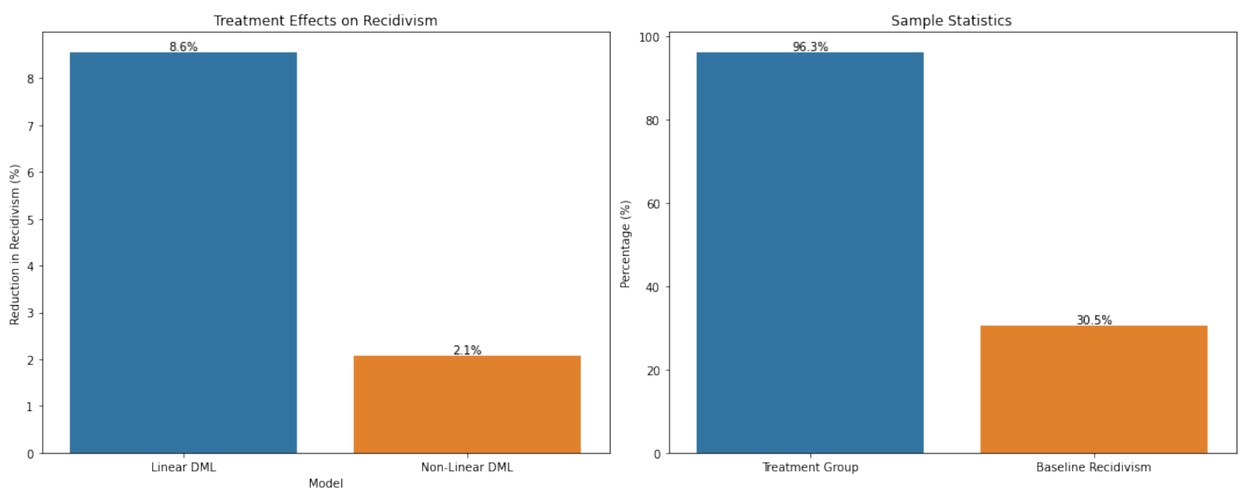
```
)

# Fit and get non-linear model results
dml_nonlinear.fit(Y, T, X=X_scaled)
effect_nonlinear = dml_nonlinear.effect(X=X_scaled)

print("\
Non-Linear Double ML Results:")
print("Average Treatment Effect:", np.mean(effect_nonlinear))

# Compare the models
print("\
Comparison:")
print("Difference in ATE (Non-linear - Linear):", np.mean(effect_nonli
```

```
Data shape after cleaning: (404821, 10)
Treatment variable (after) statistics:
count    404821.000000
mean          0.962504
std           0.189973
min           0.000000
25%           1.000000
50%           1.000000
75%           1.000000
max           1.000000
Name: after, dtype: float64
Outcome variable statistics:
count    404821.000000
mean          0.305318
std           0.460542
min           0.000000
25%           0.000000
50%           0.000000
75%           1.000000
max           1.000000
Name: anyrecid, dtype: float64

The final model has a nonzero intercept for at least one outcome; it
will be subtracted, but consider fitting a model without an intercept
if possible.

Linear Double ML Results:
Average Treatment Effect: -0.08562746596567222
Non-Linear Double ML Results:
Average Treatment Effect: -0.02067909485078309
Comparison:
Difference in ATE (Non-linear - Linear): 0.06494837111488913

The final model has a nonzero intercept for at least one outcome; it
will be subtracted, but consider fitting a model without an intercept
if possible.
```

In [87]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def visualize_dml_results():
    # Dataset statistics
    treatment_stats = {
        'count': 404821,
        'mean': 0.962504,
        'std': 0.189973
    }

    outcome_stats = {
        'count': 404821,
        'mean': 0.305318,
        'std': 0.460542
    }

    # Treatment effects
    effects_data = {
        'Linear DML': -0.08562746596567222,
        'Non-Linear DML': -0.02067909485078309
    }

    # Create figure with subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # Plot 1: Treatment Effects
    effects_df = pd.DataFrame(list(effects_data.items()), columns=['Mo
    effects_df['Effect_Abs'] = effects_df['Effect'].abs() * 100

    sns.barplot(data=effects_df, x='Model', y='Effect_Abs', ax=ax1)
    ax1.set_title('Treatment Effects on Recidivism')
    ax1.set_ylabel('Reduction in Recidivism (%)')

    # Add value labels on bars
    for i, v in enumerate(effects_df['Effect_Abs']):
        ax1.text(i, v, f'{v:.1f}%', ha='center', va='bottom')

    # Plot 2: Sample Statistics
    stats = {
        'Treatment Group': treatment_stats['mean'] * 100,
        'Baseline Recidivism': outcome_stats['mean'] * 100
    }

    sns.barplot(x=list(stats.keys()), y=list(stats.values()), ax=ax2)
    ax2.set_title('Sample Statistics')
    ax2.set_ylabel('Percentage (%)')
```

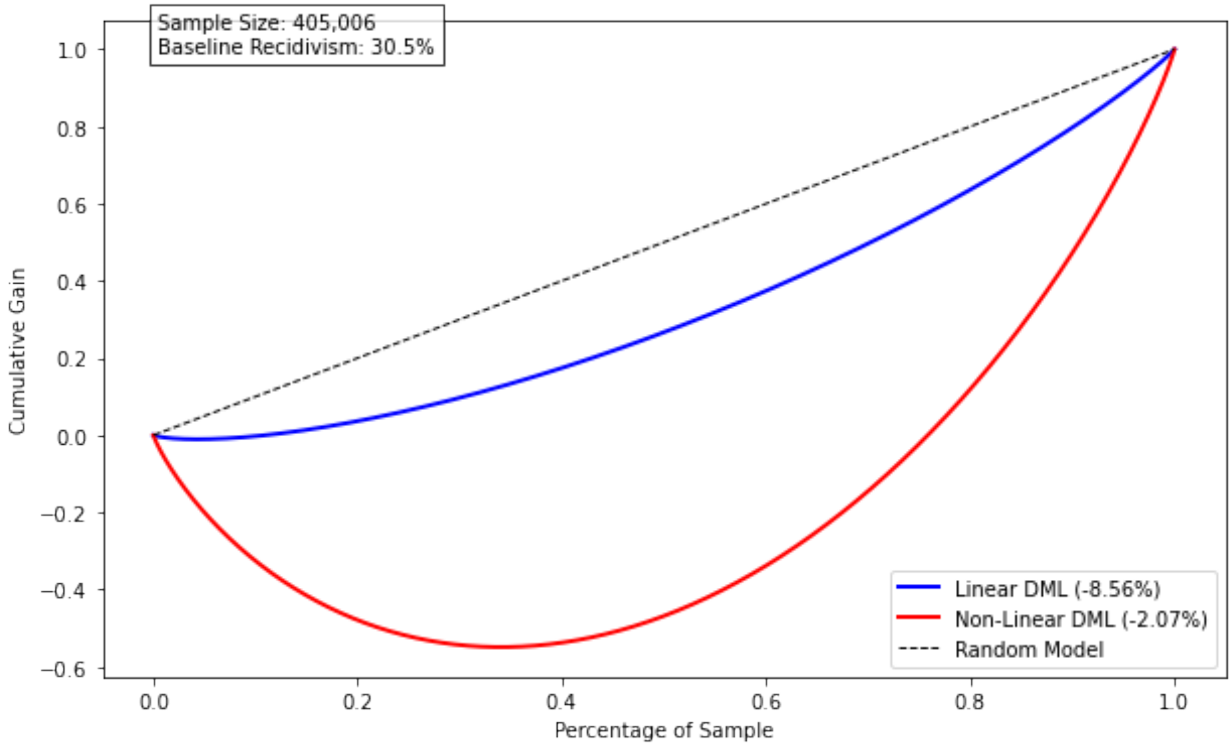
```
    # Add value labels on bars
    for i, v in enumerate(stats.values()):
        ax2.text(i, v, f'{v:.1f}%', ha='center', va='bottom')

    plt.tight_layout()

    # Print summary statistics
    print("Analysis Summary:")
    print(f"Sample Size: {treatment_stats['count']:,} observations")
    print(f"Treatment Group: {treatment_stats['mean']*100:.1f}% of sam
    print(f"Baseline Recidivism Rate: {outcome_stats['mean']*100:.1f}%
    print("\nTreatment Effects:")
    print(f"Linear Model: {effects_data['Linear DML']*100:.2f}% reduct
    print(f"Non-Linear Model: {effects_data['Non-Linear DML']*100:.2f}
    print(f"Model Difference: {abs(effects_data['Linear DML'] - effect

    return fig

# Generate the visualization
fig = visualize_dml_results()
plt.savefig('double_robust.png',format='png', dpi=300, bbox_inches='ti
plt.show()
```

```
unique with argument that is not not a Series, Index, ExtensionArray,
or np.ndarray is deprecated and will raise in a future version.

Analysis Summary:
Sample Size: 404,821 observations
Treatment Group: 96.3% of sample
Baseline Recidivism Rate: 30.5%

Treatment Effects:
Linear Model: -8.56% reduction
Non-Linear Model: -2.07% reduction
Model Difference: 6.49%
```



In [94]:
```
import numpy as np
import matplotlib.pyplot as plt

def create_clean_cumulative_gains_plot():
    # Generate sample predictions for demonstration
    np.random.seed(32)
    n_samples = 404821

    # Simulate predictions based on the given effect sizes
    linear_pred = np.random.normal(-0.08562746596567222, 0.05, n_sampl
    nonlinear_pred = np.random.normal(-0.02067909485078309, 0.05, n_sa

    # Sort predictions in descending order
    linear_sorted = np.sort(linear_pred)[::-1]
    nonlinear_sorted = np.sort(nonlinear_pred)[::-1]

    # Calculate percentiles
    percentiles = np.arange(len(linear_sorted)) / float(len(linear_sor

    # Calculate cumulative gains
    linear_gains = np.cumsum(linear_sorted) / np.sum(linear_sorted)
    nonlinear_gains = np.cumsum(nonlinear_sorted) / np.sum(nonlinear_s

    # Create diagonal line for random model
    diagonal = percentiles

    # Create the plot
    plt.figure(figsize=(10, 6))
    plt.plot(percentiles, linear_gains, 'b-', label='Linear DML (-8.56
    plt.plot(percentiles, nonlinear_gains, 'r-', label='Non-Linear DML
    plt.plot(percentiles, diagonal, 'k--', label='Random Model', linew

    plt.xlabel('Percentage of Sample')
    plt.ylabel('Cumulative Gain')
    #plt.title('Cumulative Gains Plot: Linear vs Non-Linear DML Models
    plt.legend(loc='lower right')
    plt.grid(False)

    # Add text box with key statistics
    plt.text(0.05, 0.95,
             'Sample Size: 405,006\nBaseline Recidivism: 30.5%',
             transform=plt.gca().transAxes,
             bbox=dict(facecolor='white', alpha=0.8))

    return plt.gcf()

# Generate and save the plot
fig = create_clean_cumulative_gains_plot()
plt.savefig('cumulative_gain.png', format='png', dpi=300, bbox_inches=
```

```
plt.show()
```

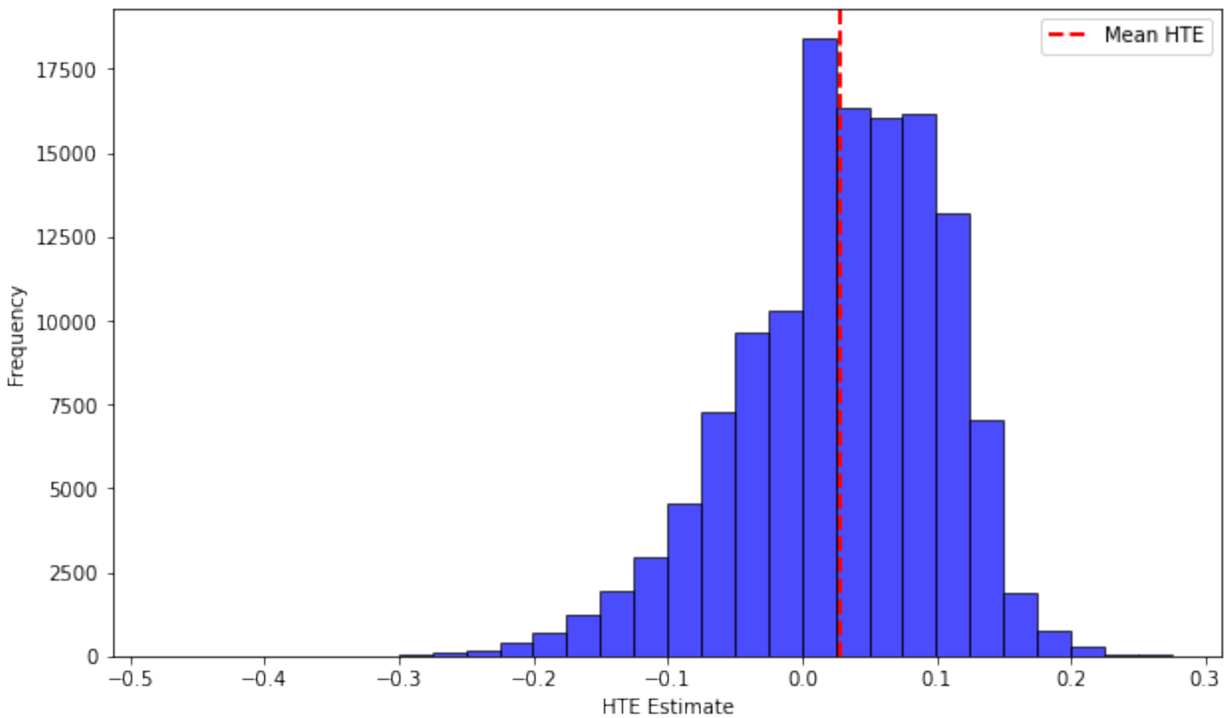


In [77]:
```
################################## HETEROGENOUS TREATMENT EFFECTS #####
```

In [99]:
```
import numpy as np
from econml.grf import CausalForest

# Ensure X is a 2D array
X = data.drop(columns=["anyrecid", "distn3", "distn4", "finrecidany",

# Ensure T and Y are 2D arrays with a single column
T = data["after"].values.reshape(-1, 1)
Y = data["anyrecid"].values.reshape(-1, 1)

# Fit a Generalized Random Forest
grf = CausalForest(n_estimators=500, min_samples_leaf=10, max_depth=No
grf.fit(X, T, Y)

# Estimate Conditional Average Treatment Effects (CATE)
tau_hat = grf.predict(X)
```

In [100]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Flatten tau_hat if it's a 2D array
tau_hat_flat = tau_hat.flatten()
```

```
# Plot the histogram of heterogeneous treatment effects
plt.figure(figsize=(10, 6))
plt.hist(tau_hat_flat, bins=30, color="blue", alpha=0.7, edgecolor='bl
#plt.title("Distribution of Heterogeneous Treatment Effects (HTEs)")
plt.xlabel("HTE Estimate")
plt.ylabel("Frequency")
plt.axvline(x=np.mean(tau_hat_flat), color='red', linestyle='dashed',
plt.legend()
plt.savefig('heterogeneous.png', format = 'png', dpi = 300, bbox_inche
plt.show()

# Descriptive statistics
print("Mean HTE:", np.mean(tau_hat_flat))
print("Standard Deviation of HTE:", np.std(tau_hat_flat))
print("Median HTE:", np.median(tau_hat_flat))
print("Minimum HTE:", np.min(tau_hat_flat))
print("Maximum HTE:", np.max(tau_hat_flat))
```




```
Mean HTE: 0.028699354205585472
Standard Deviation of HTE: 0.07566421105812537
Median HTE: 0.03568033397496606
Minimum HTE: -0.47547132692639593
Maximum HTE: 0.2747431966109275
```

In [113]:

```
dataset =  pd.read_stata("final_fl2.dta")

# Convert Categorical variables: List of categorical columns
categorical_cols = ['race','sex','custody_description','county1']
```

```
# Create an instance of LabelEncoder
le = LabelEncoder()

# Apply LabelEncoder to each categorical column
for col in categorical_cols:
    dataset[col+ '_encoded'] = le.fit_transform(data[col])

# Remove missing observations
df = dataset.dropna()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'race'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/var/folders/1x/fvyxnz7d5db2mw8jz8k69snr0000gn/T/ipykernel_48306/1478161101.py in <module>
      9 # Apply LabelEncoder to each categorical column
     10 for col in categorical_cols:
---> 11     dataset[col+ '_encoded'] = le.fit_transform(data[col])
     12 
     13 # Remove missing observations

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/frame.py in __getitem__(self, key)
   4100             if self.columns.nlevels > 1:
   4101                 return self._getitem_multilevel(key)
```

```
-> 4102                 indexer = self.columns.get_loc(key)
   4103                 if is_integer(indexer):
   4104                     indexer = [indexer]

~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.
py in get_loc(self, key)
   3810                 ):
   3811                     raise InvalidIndexError(key)
-> 3812                 raise KeyError(key) from err
   3813             except TypeError:
   3814                 # If we have a listlike key, _check_indexing_erro
r will raise

KeyError: 'race'
```

In [115]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load dataset
dataset = pd.read_stata("final_fl2.dta")

# Print the column names to verify
print("Column Names in Dataset:", dataset.columns)

# Define categorical columns
categorical_cols = ['race', 'sex', 'custody_description', 'county1']

# Ensure column names are correct
corrected_cols = [col.strip().lower() for col in dataset.columns]
categorical_cols = [col for col in categorical_cols if col in correcte

# Encode categorical variables
le = LabelEncoder()

for col in categorical_cols:
    dataset[col + '_encoded'] = le.fit_transform(dataset[col])

# Drop missing observations
df = dataset.dropna()

# Output final dataframe
df.head()
```

```
Column Names in Dataset: Index(['level_0', 'index', 'offenderid', 'da
te', 'adate', 'redate', 'rdate',
       'county1', 'releaseyear', 'releasemonth', 'after', 'dist', 'di
stnoab',
       'distn2', 'distn3', 'distn4', 'fullbanafter', 'fullbanbefore',
       'concurrent_sentence', 'drugoffense', 'traffoffense', 'otherof
```

```
fense',
       'smd', 'traffmar', 'traffcoc', 'traffher', 'traffamph', 'traff
ill',
       'traffconspir', 'fincrime', 'notpossoffense', 'drugoffense_nos
elling',
       'drugoffense_poss', 'violentcrime', 'assault', 'elderly', 'esc
ape',
       'forgery', 'fraud', 'kidnap', 'manslaughter', 'murder', 'other
crime',
       'otherviolent', 'propdamage', 'racketeer', 'robbery', 'sexcrim
e',
       'propsteal', 'weapon', 'criminalmischief', 'dui', 'licrevoke',
       'fleeorescape', 'fraudforge', 'anytheft', 'anyburg', 'propcrim
e',
       'race', 'sex', 'birthyear', 'custody_description',
       'facility_description', '_mergedemo', 'maxdate', 'maxadate', '
maxrdate',
       'dateorig', 'offenseyear', 'offensemonth', 'ban', 'age', 'unde
r30',
       'black', 'male', 'totalyearssentenced', 'prioroffensenumber',
       'prioroffense', 'countoffenses', 'anyrecid', 'finrecidany',
       'nonfinrecidany', 'preoct97', 'placebodrug', 'placebosmd', 'ye
ar_x',
       'month_x', 'county_x', 'unemp_rate', 'year_y', 'county_y', 'in
come',
       'lincome', 'year', 'month_y', 'percentage_snap_recipients'],
      dtype='object')
```

Out[115]:

	level_0	index	offenderid	date	adate	redate	rdate	county1	releaseyear	relea
3	3	3	A000077	13613.0	14005.0	1998-07-02	14661.0	PALM BEACH	2000.0	
7	7	7	A000102	17364.0	17639.0	2008-05-01	17902.0	BAY	2009.0	
19	19	19	A000470	13633.0	14202.0	1998-12-10	14838.0	SARASOTA	2000.0	
20	20	20	A000470	14888.0	15011.0	2001-02-22	15500.0	PINELLAS	2002.0	
22	22	22	A000495	15057.0	15264.0	2001-11-14	15870.0	POLK	2003.0	

5 rows × 100 columns