Data Models

Import Libraries

In [86]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

from sklearn.metrics import r2_score
import statsmodels.api as sm
from statsmodels.api import OLS
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import KFold
from sklearn.decomposition import PCA

#import pydotplus
#import io
from sklearn.tree import export_graphviz
from IPython.display import Image
from IPython.display import display
import seaborn as sns
%matplotlib inline
```

Define Functions

In [87]:
```
def despine():
    sns.despine(left=True, bottom=True)

def get_axs(rows, columns, fig_size_width, fig_size_height):
    dims = (fig_size_width, fig_size_height)
    fig, axs = plt.subplots(rows, columns, figsize=dims)
    if(rows*columns>1):
         axs = axs.ravel()
    return axs

def get_accuracy_model(X, Y, model):
    Y_pred = model.predict(X)
    misclassification_rate = np.mean([int(x) for x in Y_pred != Y])
    return 1 - misclassification_rate

def get_accuracy_pred(Y, Y_pred):
    misclassification_rate = np.mean([int(x) for x in Y_pred != Y])
    return 1 - misclassification_rate

def split_dataset(data, train_size_pc, y_col):
    np.random.seed(9001)
    msk = np.random.rand(len(data)) < train_size_pc
    data_train = data[msk]
    data_test = data[~msk]

    x_train = data_train.iloc[:,0:y_col]
    y_train = data_train.iloc[:,y_col]

    x_test = data_test.iloc[:,0:y_col]
    y_test = data_test.iloc[:,y_col]
    return x_train, y_train, x_test, y_test

def set_title_xlabel_ylabel(ax, title, xlabel, ylabel):
    ax.set_title(title)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
```

In [88]:
```
sns.set(rc={'axes.facecolor':'white', 'figure.facecolor':'white'})
sns.set_style("whitegrid")
sns.set(font_scale=1.3)
```

Import Dataset

In [89]:
```
census_data = pd.read_csv("crime_data.csv", index_col=0)
results = pd.DataFrame([], columns = ["model", "train_score", "test_score"])
```

Dropping all rows with missing values

In [90]:
```
census_data = census_data.dropna(how='any')
```

Hot One Encoding Categorical Variables

In [91]:
```
#categorical
cat_vars = ['year']

split = {}
split_test = {}

def hot_one_encoding(data, cat_vars):
    for var in cat_vars:
        s_var = pd.Series(data[var])
        split[var] = pd.get_dummies(s_var)

        func = lambda x: var + '_'+ str(x)

        cols = list(map(func, list(split[var].columns)[1:]))
        split[var] = split[var].drop(split[var].columns[0], axis=1)
        split[var].columns = cols

        data = data.join(split[var])

        del data[var]
    return data
```

Normalizing all quantitative variables

In [92]:
```
quant_vars = ['msa', 'pop', 'r1', 'r2', 'r3', 'r4', 'r5', 'r6', 'r7', 'm1', 'm
2',
       'm3', 'm4', 'm5', 'i1', 'i2', 'e1', 'e2', 'e3', 'e4', 'e5', 'a1', 'a2',
       'a3', 'a4', 'a5', 'a6', 'a7', 'e6', 'vr', 'mtof', 'firearms',
       'murder_rate']
```

In [93]:
```
census_data = census_data.drop(['msa', 'year'], axis=1)
# census_data = hot_one_encoding(census_data, cat_vars)
quant_vars = list(census_data.columns)
quant_vars = ['pop', 'i1', 'i2', 'firearms']
for var in quant_vars:
    var_mean = np.mean(census_data[var])
    var_std = np.std(census_data[var])
    census_data[var + '_std'] = (census_data[var]-var_mean)/var_std
    del census_data[var]
```

Train and Test Split

In [94]:
```
np.random.seed(9001)
msk = np.random.rand(len(census_data)) < 0.75
census_train = census_data[msk]
census_test = census_data[~msk]
features = list(census_data.columns)
features.remove('murder_rate')
end = len(census_data.columns)
x_train = census_train[features]
y_train = census_train['murder_rate']

x_test = census_test[features]
y_test = census_test['murder_rate']
```

Baseline Model

In [95]:
```
#LINEAR REGRESSION
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
y_pred_train = lin_reg.predict(x_train)
y_pred_test = lin_reg.predict(x_test)
```

In [96]:
```
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Linear Regression", "train_score":train_sco
re, "test_score":test_score}, ignore_index=True)
```

In [97]:

```
x_train_with_constants = sm.add_constant(x_train)
est = sm.OLS(y_train, x_train_with_constants)
est = est.fit()
print(est.summary())
```

```
                            OLS Regression Results
==============================================================================
===
Dep. Variable:            murder_rate   R-squared:                       0.
516
Model:                            OLS   Adj. R-squared:                  0.
511
Method:                 Least Squares   F-statistic:                     9
2.87
Date:                Thu, 07 Dec 2017   Prob (F-statistic):
0.00
Time:                        20:21:27   Log-Likelihood:                -635
9.6
No. Observations:                2642   AIC:                         1.278e
+04
Df Residuals:                    2611   BIC:                         1.296e
+04
Df Model:                          30

Covariance Type:            nonrobust

==============================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const          7.2141      4.847      1.488      0.137      -2.290       1
6.718
r1             2.3092      1.743      1.325      0.185      -1.109
5.727
r2            20.6499      1.731     11.928      0.000      17.255       2
4.045
r3             6.0993      2.646      2.305      0.021       0.911       1
1.287
r4             1.6527      3.724      0.444      0.657      -5.649
8.955
r5           -96.1706     12.809     -7.508      0.000    -121.288      -7
1.054
r6            19.7812      2.101      9.416      0.000      15.662       2
3.901
r7            52.8924      4.096     12.913      0.000      44.860       6
0.924
m1           165.8571     99.448      1.668      0.095     -29.147      36
0.861
m2           205.8156     99.645      2.065      0.039      10.424      40
1.207
m3           198.8532     99.550      1.998      0.046       3.648      39
4.058
m4           158.9648     99.878      1.592      0.112     -36.883      35
4.812
m5           185.9452     99.524      1.868      0.062      -9.208      38
1.099
e1            13.5237      4.084      3.311      0.001       5.515       2
1.532
e2            -4.8460      5.232     -0.926      0.354     -15.106
```

```
5.414
e3               6.4311      3.362      1.913      0.056      -0.162       1
3.025
e4              15.9294      4.698      3.391      0.001       6.718       2
5.141
e5               2.3189      1.673      1.386      0.166      -0.961
5.599
a1            -203.0996    100.051     -2.030      0.042    -399.288       -
6.912
a2            -216.1023     99.691     -2.168      0.030    -411.584      -2
0.620
a3            -168.8144    100.089     -1.687      0.092    -365.076       2
7.447
a4            -214.3905    100.235     -2.139      0.033    -410.938      -1
7.843
a5            -183.7737     99.897     -1.840      0.066    -379.658       1
2.111
a6            -192.4230     99.743     -1.929      0.054    -388.007
3.161
a7            -191.6246     99.502     -1.926      0.054    -386.735
3.486
e6              -1.6722      3.369     -0.496      0.620      -8.278
4.934
vr               2.3909      1.185      2.018      0.044       0.068
4.714
mtof            -1.1853      1.708     -0.694      0.488      -4.534
2.163
pop_std         -0.1005      0.070     -1.435      0.151      -0.238
0.037
i1_std          -0.7292      0.338     -2.157      0.031      -1.392       -
0.066
i2_std          -0.0210      0.373     -0.056      0.955      -0.753
0.711
firearms_std    -0.0931      0.060     -1.559      0.119      -0.210
0.024
===========================================================================
===
Omnibus:                     1131.306   Durbin-Watson:                   2.
000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            12808.
311
Skew:                           1.712   Prob(JB):
0.00
Kurtosis:                      13.229   Cond. No.                     5.74e
+15
===========================================================================
===

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corr
ectly specified.
[2] The smallest eigenvalue is 2.54e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Multiple Models - Ridge, Lasso and Polynomial

Ridge with Cross Validation

In [98]:
```
ridge = RidgeCV()
ridge.fit(x_train, y_train)
y_pred_train = ridge.predict(x_train)
y_pred_test = ridge.predict(x_test)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Ridge Cross Validated", "train_score":train
_score, "test_score":test_score}, ignore_index=True)
```

In [99]:
```
RidgeCoefficients = pd.DataFrame(list(zip([np.abs(coef) for coef in ridge.coef
_], [np.abs(coef)/coef for coef in ridge.coef_], x_train.columns)), columns=[
'Value', 'Sign', 'Coef'])
RidgeCoefficients = RidgeCoefficients[RidgeCoefficients['Value']>0.1].sort_val
ues(['Value'], ascending=[0])
RidgeCoefficients.index = list(range(0, RidgeCoefficients.shape[0]))
RidgeCoefficients
```

Out[99]:

	Value	Sign	Coef
0	31.491006	1.0	r7
1	26.751318	-1.0	r5
2	20.278714	-1.0	m1
3	20.041086	-1.0	a2
4	14.669455	-1.0	r4
5	12.645540	1.0	e1
6	12.203959	1.0	e4
7	11.208201	1.0	r6
8	10.548848	1.0	m3
9	10.152961	1.0	r2
10	9.419112	1.0	a3
11	7.982711	-1.0	r1
12	7.676144	1.0	m2
13	7.096209	-1.0	m4
14	7.065352	-1.0	a1
15	5.859527	-1.0	e2
16	4.330672	1.0	a5
17	4.271342	1.0	e3
18	3.448684	-1.0	r3
19	3.047561	-1.0	e6
20	2.819151	-1.0	a4
21	2.429502	1.0	a6
22	2.334499	-1.0	m5
23	2.183503	1.0	e5
24	1.573345	1.0	vr
25	1.434026	-1.0	mtof
26	0.873549	-1.0	i1_std
27	0.719188	1.0	a7
28	0.143997	1.0	i2_std
29	0.111040	-1.0	firearms_std

Lasso with Cross Validation

In [100]:

```
lasso = LassoCV()
lasso.fit(x_train, y_train)
y_pred_train = lasso.predict(x_train)
y_pred_test = lasso.predict(x_test)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Lasso Cross Validated", "train_score":train
_score, "test_score":test_score}, ignore_index=True)
```

In [101]:
```
LassoCoefficients = pd.DataFrame(list(zip([np.abs(coef) for coef in lasso.coef_], [np.abs(coef)/coef for coef in lasso.coef_], x_train.columns)), columns=['Value', 'Sign', 'Coef'])
LassoCoefficients = LassoCoefficients[LassoCoefficients['Value']>0.1].sort_values(['Value'], ascending=[0])
LassoCoefficients.index = list(range(0, LassoCoefficients.shape[0]))
LassoCoefficients
```

```
C:\Software\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered in double_scalars
  """Entry point for launching an IPython kernel.
```

Out[101]:

	Value	Sign	Coef
0	35.319289	1.0	r7
1	26.304059	-1.0	a2
2	19.096901	-1.0	m1
3	15.198220	1.0	r6
4	14.014587	1.0	r2
5	10.903004	-1.0	r4
6	8.586073	1.0	e4
7	7.662272	1.0	e1
8	4.758945	1.0	m3
9	4.409728	-1.0	r5
10	4.341127	-1.0	e6
11	3.418771	-1.0	r1
12	1.292123	1.0	vr
13	1.289610	-1.0	i1_std
14	0.569240	1.0	i2_std
15	0.568501	1.0	e5
16	0.439553	1.0	e3
17	0.263765	-1.0	mtof
18	0.126567	-1.0	firearms_std

Polynomial Features - Linear, Ridge and Lasso

In [102]:
```
poly = PolynomialFeatures(degree = 2)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)

lin_reg = LinearRegression()
ridge = RidgeCV()
lasso = LassoCV()

lin_reg.fit(x_train_poly, y_train)
y_pred_train = lin_reg.predict(x_train_poly)
y_pred_test = lin_reg.predict(x_test_poly)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Linear Regression with Polynomial Features"
, "train_score":train_score, "test_score":test_score}, ignore_index=True)

ridge.fit(x_train_poly, y_train)
y_pred_train = ridge.predict(x_train_poly)
y_pred_test = ridge.predict(x_test_poly)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Lasso Cross Validated with Polynomial Featu
res", "train_score":train_score, "test_score":test_score}, ignore_index=True)

lasso.fit(x_train_poly, y_train)
y_pred_train = lasso.predict(x_train_poly)
y_pred_test = lasso.predict(x_test_poly)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Ridge Cross Validated with Polynomial Featu
res", "train_score":train_score, "test_score":test_score}, ignore_index=True)
```

```
C:\Software\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_desce
nt.py:491: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Fitting data with very small alpha may cau
se precision problems.
  ConvergenceWarning)
C:\Software\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_desce
nt.py:491: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Fitting data with very small alpha may cau
se precision problems.
  ConvergenceWarning)
```

Interaction Terms

From our EDA we hypothesized that multi-ethnic communnities might have higher murder rates. To test this hypothesis we will use a interaction term that multiplies all the race proportions and add to our train and test models

In [103]:
```
x_train['multi_ethnic'] = x_train['r1'] * x_train['r2'] * x_train['r3'] * x_tr
ain['r4'] * x_train['r5'] * x_train['r6'] * x_train['r7']
x_test['multi_ethnic'] = x_test['r1'] * x_test['r2'] * x_test['r3'] * x_test[
'r4'] * x_test['r5'] * x_test['r6'] * x_test['r7']
```

Normalizing

In [104]:
```
quant_vars = ['multi_ethnic']
for var in quant_vars:
    var_mean = np.mean(x_train[var])
    var_std = np.std(x_train[var])
    x_train[var + '_std'] = (x_train[var]-var_mean)/var_std
    del x_train[var]
for var in quant_vars:
    var_mean = np.mean(x_test[var])
    var_std = np.std(x_test[var])
    x_test[var + '_std'] = (x_test[var]-var_mean)/var_std
    del x_test[var]
```

Modeling

In [105]:
```
lin_reg.fit(x_train, y_train)
y_pred_train = lin_reg.predict(x_train)
y_pred_test = lin_reg.predict(x_test)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Linear Regression with Interaction", "train
_score":train_score, "test_score":test_score}, ignore_index=True)

ridge.fit(x_train, y_train)
y_pred_train = ridge.predict(x_train)
y_pred_test = ridge.predict(x_test)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Lasso Cross Validated with Interaction", "t
rain_score":train_score, "test_score":test_score}, ignore_index=True)

lasso.fit(x_train, y_train)
y_pred_train = lasso.predict(x_train)
y_pred_test = lasso.predict(x_test)
train_score = r2_score(y_train, y_pred_train)
test_score = r2_score(y_test, y_pred_test)
results = results.append({"model":"Ridge Cross Validated with Interaction", "t
rain_score":train_score, "test_score":test_score}, ignore_index=True)
```

Checking for Significance

In [106]:

```
x_train_with_constants = sm.add_constant(x_train)
est = sm.OLS(y_train, x_train_with_constants)
est = est.fit()
print(est.summary())
```

```
                            OLS Regression Results                            
===========================================================================
===
Dep. Variable:            murder_rate   R-squared:                       0.
517
Model:                            OLS   Adj. R-squared:                  0.
512
Method:                 Least Squares   F-statistic:                     9
0.20
Date:                Thu, 07 Dec 2017   Prob (F-statistic):               
0.00
Time:                        20:21:32   Log-Likelihood:                 -635
6.8
No. Observations:                2642   AIC:                          1.278e
+04
Df Residuals:                    2610   BIC:                          1.297e
+04
Df Model:                          31                                         

Covariance Type:            nonrobust                                         

===========================================================================
=========
                       coef    std err          t      P>|t|      [0.025      
   0.975]
---------------------------------------------------------------------------
---------
const                8.4276      4.871      1.730      0.084      -1.123      
   17.978
r1                   2.5337      1.744      1.453      0.146      -0.886      
    5.954
r2                  20.8673      1.732     12.046      0.000      17.470      
   24.264
r3                   5.9533      2.644      2.251      0.024       0.768      
   11.138
r4                  -0.2868      3.813     -0.075      0.940      -7.763      
    7.189
r5                 -92.6253     12.888     -7.187      0.000    -117.898      
  -67.353
r6                  19.8190      2.099      9.442      0.000      15.703      
   23.935
r7                  52.1664      4.105     12.709      0.000      44.118      
   60.215
m1                 168.1788     99.368      1.692      0.091     -26.670      
  363.028
m2                 208.1425     99.566      2.090      0.037      12.906      
  403.379
m3                 200.5634     99.469      2.016      0.044       5.518      
  395.609
m4                 159.7407     99.794      1.601      0.110     -35.942      
  355.423
m5                 188.2398     99.444      1.893      0.058      -6.758      
  383.238
e1                  12.5317      4.103      3.054      0.002       4.487      
   20.577
e2                  -5.6044      5.238     -1.070      0.285     -15.876      
```

```
    4.667
e3                    5.2656       3.397       1.550       0.121       -1.395
   11.926
e4                   14.9219       4.714       3.166       0.002        5.679
   24.165
e5                    2.3612       1.671       1.413       0.158       -0.916
    5.639
a1                 -206.4531      99.977      -2.065       0.039     -402.495
  -10.411
a2                 -218.8110      99.614      -2.197       0.028     -414.141
  -23.481
a3                 -172.0718     100.014      -1.720       0.085     -368.187
   24.043
a4                 -217.1128     100.157      -2.168       0.030     -413.507
  -20.718
a5                 -186.4847      99.819      -1.868       0.062     -382.217
    9.248
a6                 -195.0243      99.665      -1.957       0.050     -390.455
    0.406
a7                 -193.8795      99.422      -1.950       0.051     -388.834
    1.075
e6                   -3.0429       3.417      -0.890       0.373       -9.744
    3.658
vr                    2.2302       1.186       1.881       0.060       -0.095
    4.555
mtof                 -1.0897       1.707      -0.638       0.523       -4.437
    2.257
pop_std              -0.0894       0.070      -1.275       0.202       -0.227
    0.048
i1_std               -0.7584       0.338      -2.244       0.025       -1.421
   -0.096
i2_std               -0.0205       0.373      -0.055       0.956       -0.751
    0.710
firearms_std         -0.0881       0.060      -1.477       0.140       -0.205
    0.029
multi_ethnic_std      0.1518       0.065       2.329       0.020        0.024
    0.280
============================================================================
===
Omnibus:                     1140.844   Durbin-Watson:                     1.
992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              13053.
873
Skew:                           1.727   Prob(JB):
0.00
Kurtosis:                      13.327   Cond. No.                       4.50e
+15
============================================================================
===

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corr
ectly specified.
[2] The smallest eigenvalue is 4.14e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Modelling Results

In [107]:
```
results.index = results.model
```

In [108]:
```
results.drop(['model'], axis=1)
```

Out[108]:

	train_score	test_score
model		
Linear Regression	0.516235	0.506611
Ridge Cross Validated	0.507715	0.491949
Lasso Cross Validated	0.500520	0.485036
Linear Regression with Polynomial Features	0.773114	0.546192
Lasso Cross Validated with Polynomial Features	0.662992	0.647014
Ridge Cross Validated with Polynomial Features	0.626490	0.618030
Linear Regression with Interaction	0.517228	0.507337
Lasso Cross Validated with Interaction	0.509099	0.493297
Ridge Cross Validated with Interaction	0.504211	0.489468