Predicting determinants of crimes in US Communities

This project uses the communities dataset from UCI learning repository to explain factors responsible for crimes in many US communities. The data contains 2215 instances with 147 records. We have carefully selected about 20 variables which we feel can trigger crimes in the communities.

Some variables are constructed in different ways to essentially measure the same thing. For instance, reporting number of people under poverty and percentage of people living under poverty, the 2 variables essentially are quantifying thesame thing, in such situations, we stick to only 1 of the variables, preferably percentage measures.

Importing the necessary libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Import the dataset

In [2]:

```
data = pd.read_csv("/content/drive/MyDrive/Crime Analytics/crimenew.csv")
data.head()
```

Out[2]:

	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t29	pctWWage	pctWSocSec	pctWPubAsst	perCapInc	...	MalePctNevMa
0	2.57	0.18	98.48	0.84	1.66	24.90	77.02	35.90	5.43	16201	...	31.
1	2.40	6.50	91.93	1.13	1.17	27.58	79.77	27.82	6.00	15138	...	33.
2	2.92	1.41	96.01	1.76	3.61	27.30	89.65	15.44	3.02	17642	...	28.
3	2.59	1.58	90.57	7.00	5.08	23.23	81.68	31.65	2.30	19643	...	30.
4	2.38	0.42	99.19	0.24	0.52	23.42	66.11	44.02	6.67	10246	...	26.

5 rows × 29 columns

In [3]:

```
data.tail()
```

Out[3]:

	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t29	pctWWage	pctWSocSec	pctWPubAsst	perCapInc	...	MalePctNe
2210	2.76	0.82	97.60	1.26	1.86	24.96	78.41	30.55	2.55	26895	...	
2211	2.50	7.58	81.61	1.79	16.27	30.11	74.42	31.61	9.08	14715	...	
2212	2.72	6.57	86.75	3.94	7.69	28.95	83.63	23.55	3.98	19300	...	
2213	2.63	2.13	93.26	3.49	4.42	22.08	80.52	28.24	2.38	46070	...	
2214	2.65	1.52	96.79	1.03	2.31	24.75	82.47	28.50	3.14	19099	...	

5 rows × 29 columns

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2215 entries, 0 to 2214
Data columns (total 29 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   householdsize       2215 non-null   float64
 1   racepctblack        2215 non-null   float64
 2   racePctWhite        2215 non-null   float64
 3   racePctAsian        2215 non-null   float64
 4   racePctHisp         2215 non-null   float64
 5   agePct12t29         2215 non-null   float64
 6   pctWWage            2215 non-null   float64
 7   pctWSocSec          2215 non-null   float64
 8   pctWPubAsst         2215 non-null   float64
 9   perCapInc           2215 non-null   int64  
 10  whitePerCap         2215 non-null   int64  
 11  blackPerCap         2215 non-null   int64  
```

```
 12  indianPerCap         2215 non-null   int64
 13  AsianPerCap          2215 non-null   int64
 14  OtherPerCap          2214 non-null   float64
 15  HispPerCap           2215 non-null   int64
 16  PctPopUnderPov       2215 non-null   float64
 17  PctNotHSGrad         2215 non-null   float64
 18  PctUnemployed        2215 non-null   float64
 19  MalePctNevMarr       2215 non-null   float64
 20  TotalPctDiv          2215 non-null   float64
 21  PctKidsBornNeverMar  2215 non-null   float64
 22  PctImmigRec5         2215 non-null   float64
 23  PctPersDenseHous     2215 non-null   float64
 24  MedRentPctHousInc    2215 non-null   float64
 25  PctSameCity85        2215 non-null   float64
 26  PopDens              2215 non-null   float64
 27  ViolentCrimesPerPop  2215 non-null   object
 28  nonViolPerPop        2215 non-null   object
dtypes: float64(21), int64(6), object(2)
memory usage: 502.0+ KB
```

Convert Violent and nonViolent Crimes to int

In [5]:

```
data[['ViolentCrimesPerPop','nonViolPerPop']] = data[['ViolentCrimesPerPop','nonViolPerPop']].apply(pd.to_numeric,errors='coerce')
```

In [6]:

```
data.isnull().sum()
```

Out[6]:

```
householdsize            0
racepctblack             0
racePctWhite             0
racePctAsian             0
racePctHisp              0
agePct12t29              0
pctWWage                 0
pctWSocSec               0
pctWPubAsst              0
perCapInc                0
whitePerCap              0
blackPerCap              0
indianPerCap             0
AsianPerCap              0
OtherPerCap              1
HispPerCap               0
PctPopUnderPov           0
PctNotHSGrad             0
PctUnemployed            0
MalePctNevMarr           0
TotalPctDiv              0
PctKidsBornNeverMar      0
PctImmigRec5             0
PctPersDenseHous         0
MedRentPctHousInc        0
PctSameCity85            0
PopDens                  0
ViolentCrimesPerPop    221
nonViolPerPop           97
dtype: int64
```

In [7]:

```
# Remove missing values

df = data.dropna()
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
householdsize          0
racepctblack           0
racePctWhite           0
racePctAsian           0
racePctHisp            0
agePct12t29            0
pctWWage               0
pctWSocSec             0
pctWPubAsst            0
perCapInc              0
whitePerCap            0
blackPerCap            0
indianPerCap           0
AsianPerCap            0
OtherPerCap            0
HispPerCap             0
PctPopUnderPov         0
PctNotHSGrad           0
```

```
PctUnemployed          0
MalePctNevMarr         0
TotalPctDiv            0
PctKidsBornNeverMar    0
PctImmigRec5           0
PctPersDenseHous       0
MedRentPctHousInc      0
PctSameCity85          0
PopDens                0
ViolentCrimesPerPop    0
nonViolPerPop          0
dtype: int64
```

In [9]:

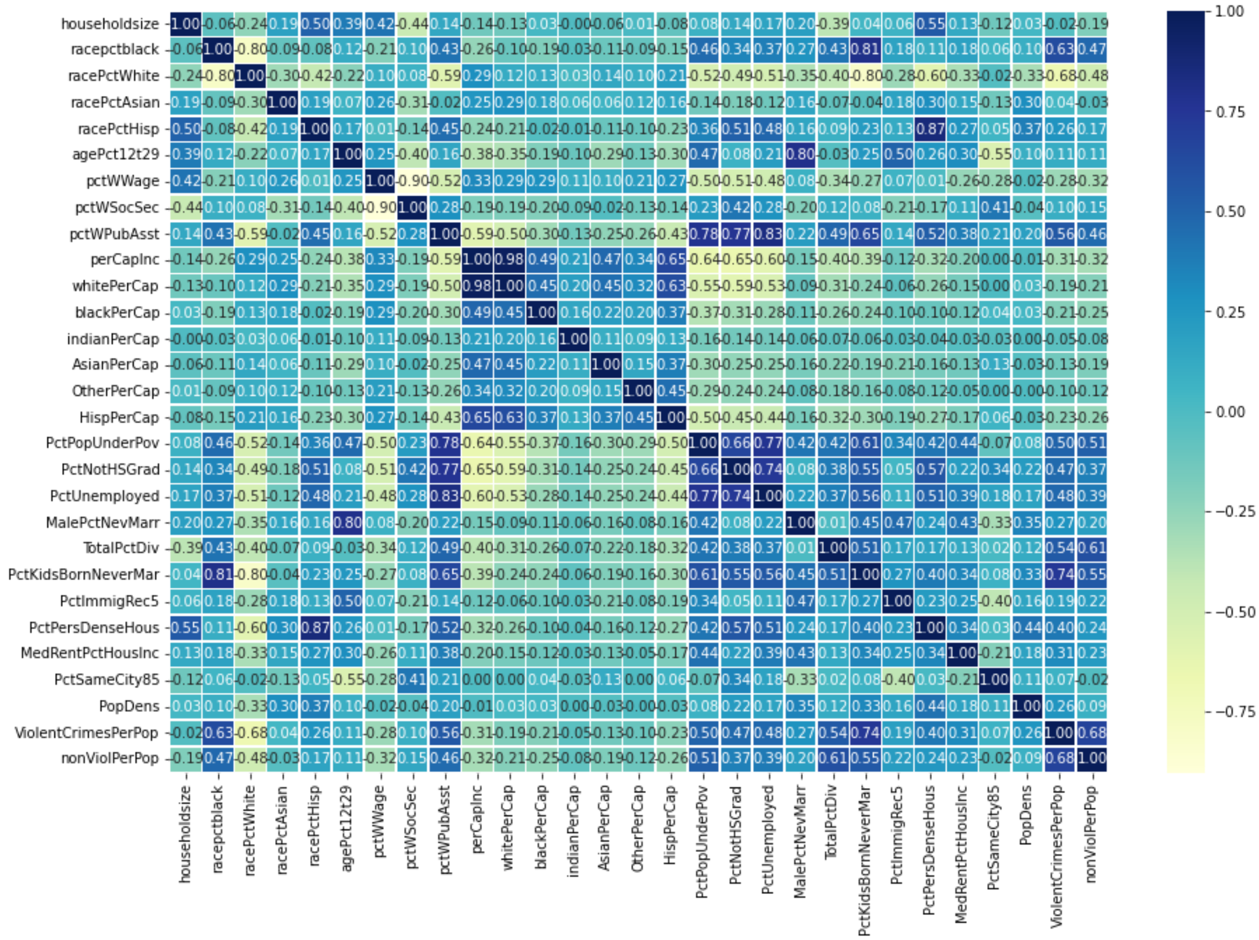
```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
householdsize	1901.0	2.712167	0.347454	1.60	2.50	2.66	2.86	5.28
racepctblack	1901.0	9.358958	13.935927	0.00	0.93	3.04	11.43	96.67
racePctWhite	1901.0	83.466423	16.357057	2.68	75.77	89.61	95.96	99.63
racePctAsian	1901.0	2.822799	4.738172	0.06	0.63	1.27	2.88	57.46
racePctHisp	1901.0	8.718985	15.449951	0.12	0.95	2.43	8.92	95.29
agePct12t29	1901.0	27.601094	6.153583	9.38	24.37	26.78	29.20	70.51
pctWWage	1901.0	78.189863	7.841396	31.68	73.45	78.55	83.76	96.62
pctWSocSec	1901.0	26.577880	8.252830	4.81	20.90	26.66	31.72	76.39
pctWPubAsst	1901.0	6.755381	4.482357	0.50	3.36	5.62	9.09	26.92
perCapInc	1901.0	15604.296160	6289.037905	5237.00	11563.00	14087.00	17910.00	63302.00
whitePerCap	1901.0	16616.188322	6381.751863	5472.00	12643.00	15087.00	18710.00	68850.00
blackPerCap	1901.0	11582.885324	9374.183691	0.00	6748.00	9784.00	14549.00	212120.00
indianPerCap	1901.0	12328.749079	15519.246128	0.00	6405.00	9943.00	14807.00	480000.00
AsianPerCap	1901.0	14293.126775	9627.835796	0.00	8542.00	12393.00	17351.00	106165.00
OtherPerCap	1901.0	9480.354024	8070.968529	0.00	5615.00	8205.00	11471.00	137000.00
HispPerCap	1901.0	11036.994740	5774.773908	0.00	7288.00	9709.00	13431.00	54648.00
PctPopUnderPov	1901.0	11.664277	8.467334	0.64	4.63	9.38	17.04	48.82
PctNotHSGrad	1901.0	22.656597	11.079400	2.09	14.16	21.54	29.59	73.66
PctUnemployed	1901.0	6.009932	2.705395	1.32	4.09	5.47	7.41	23.83
MalePctNevMarr	1901.0	30.654561	8.045644	12.06	25.45	29.02	33.44	76.32
TotalPctDiv	1901.0	10.867307	3.016394	2.83	8.59	11.03	13.08	19.11
PctKidsBornNeverMar	1901.0	3.109295	3.058334	0.00	1.07	2.06	3.93	24.19
PctImmigRec5	1901.0	20.784808	12.271561	0.00	11.72	19.74	27.63	76.16
PctPersDenseHous	1901.0	4.400295	5.939846	0.15	1.31	2.49	4.96	59.49
MedRentPctHousInc	1901.0	26.352814	2.912996	14.90	24.40	26.20	28.10	35.10
PctSameCity85	1901.0	76.990142	10.837694	27.95	71.74	79.13	84.67	96.59
PopDens	1901.0	2804.223461	2945.490095	10.00	1175.60	2003.50	3278.30	44229.90
ViolentCrimesPerPop	1901.0	583.712941	608.430184	6.64	163.75	369.30	792.93	4877.06
nonViolPerPop	1901.0	4941.049011	2786.789842	116.79	2913.24	4479.11	6265.54	27119.76

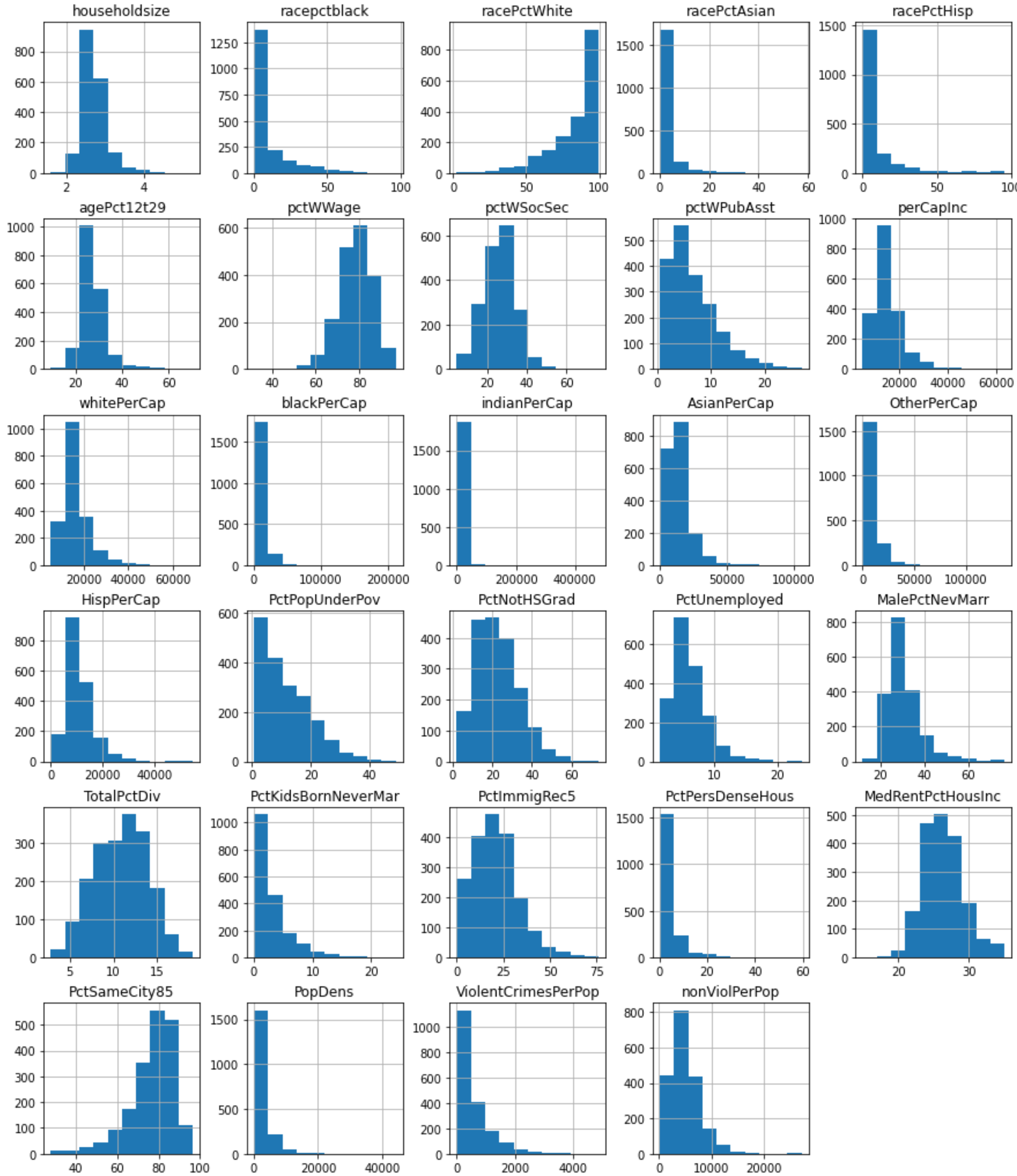
In [10]:

```
corr_matrix = df.corr()
plt.figure(figsize=(15,10))
sns.heatmap(corr_matrix,
            annot=True,
            linewidths=0.5,
            fmt='.2f',
            cmap='YlGnBu');
```



In [11]:

```
def show_hist(x):
    plt.rcParams["figure.figsize"] = 15,18
    x.hist()
show_hist(df)
```



Log Transform the target variables--ViolentCrimesPerPop and nonViolPerPop

In [12]:

```
df['Ln_VCrime'] = np.log(df['ViolentCrimesPerPop'])
df['Ln_nVCrime'] = np.log(df['nonViolPerPop'])
```

Drop Violent and nonViolent Crimes variables

In [13]:

```
df.drop(['ViolentCrimesPerPop','nonViolPerPop'],axis=1,inplace=True)
```

Perform a Baseline algorithm test

In [14]:

```
from sklearn.model_selection import train_test_split
```

In [15]:

```
X = df.drop(['Ln_VCrime','Ln_nVCrime'],axis=1)
y = df['Ln_nVCrime']
```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

Import the algorithms

In [17]:
```
from sklearn.linear_model import LinearRegression,Lasso,ElasticNetCV,ElasticNet
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

We standard the features because some of them are in percentage while some are in real numbers

In [18]:
```
pipelines =[]
pipelines.append(('ScalerLR',Pipeline([('Scaler',StandardScaler()),('Lr',LinearRegression())])))
pipelines.append(('ScalerLasso',Pipeline([('Scaler',StandardScaler()),('LASSO',Lasso())])))
pipelines.append(('ScalerEN',Pipeline([('Scaler',StandardScaler()),('EN',ElasticNet())])))
pipelines.append(('ScalerKnn',Pipeline([('Scaler',StandardScaler()),('KNN',KNeighborsRegressor())])))
pipelines.append(('ScalerRf',Pipeline([('Scaler',StandardScaler()),('RF',RandomForestRegressor())])))

results = []
names = []

for name,model in pipelines:
  kfold = KFold(n_splits=10)
  cv_results = cross_val_score(model,X_train,y_train,cv=kfold,scoring='neg_mean_squared_error')
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name,cv_results.mean(),cv_results.std())
  print(msg)
```

```
ScalerLR: -0.154098 (0.038381)
ScalerLasso: -0.335276 (0.048884)
ScalerEN: -0.335276 (0.048884)
ScalerKnn: -0.172971 (0.037560)
ScalerRf: -0.152467 (0.043657)
```

RandomForest Regressor seems to perform better than all the other algorithms, hence we focus on using Randomforest to build the final model

In [19]:
```
from sklearn.model_selection import GridSearchCV
```

In [20]:
```
from sklearn.feature_selection import SelectKBest,f_regression, VarianceThreshold
```

In [21]:
```
pipe = Pipeline([("std",StandardScaler()),
                 ("var",VarianceThreshold()),
                 ("selector",SelectKBest()),
                 ("regressor",RandomForestRegressor())])

params = [
          {"selector__k":[6,7,8,9,10,11,12,13,14]},
          {"regressor":[RandomForestRegressor()],
           "regressor__n_estimators":[10,100,200,1000],
          "regressor__max_features":['auto','sqrt','log2'],
           "regressor__max_depth":[2,3,4,5,6,7]}
]
```

In [22]:
```
grid = GridSearchCV(pipe,params,scoring='neg_mean_squared_error',cv=5)
Model_result = grid.fit(X_train, y_train)
```

In [23]:
```
best_est = Model_result.best_estimator_
print(best_est)
```

```
Pipeline(steps=[('std', StandardScaler()), ('var', VarianceThreshold()),
                ('selector', SelectKBest(k=14)),
                ('regressor', RandomForestRegressor())])
```

In [24]:
```
best_est.score(X_test,y_test)
```

Out[24]:
```
0.5798138375912989
```

In [26]:
```
y_pred = Model_result.predict(X_test)
y_pred_tr = best_est.predict(X_train)
```

```
r_square = r2_score(y_train,y_pred_tr)
r_square
```

Out[26]:

```
0.9323748441153882
```

In [27]:

```
r_square_test = r2_score(y_test,y_pred)
r_square_test
```

Out[27]:

```
0.5798138375912989
```

In [28]:

```
col_after_var = X_train.columns[best_est['var'].get_support()]
mask_sel = best_est['selector'].get_support(indices=True)
final_feature_cols = col_after_var[mask_sel]
```

In [29]:

```
final_feature_cols
```

Out[29]:

```
Index(['racepctblack', 'racePctWhite', 'racePctAsian', 'racePctHisp',
       'agePct12t29', 'pctWPubAsst', 'perCapInc', 'whitePerCap',
       'PctPopUnderPov', 'MalePctNevMarr', 'TotalPctDiv',
       'PctKidsBornNeverMar', 'PctPersDenseHous', 'PopDens'],
      dtype='object')
```

In [30]:

```
coef = Model_result.best_estimator_.named_steps['regressor'].feature_importances_
importance = np.abs(coef)
importance
```

Out[30]:

```
array([0.04015851, 0.05860437, 0.0256236 , 0.03619961, 0.04108656,
       0.030938  , 0.0201021 , 0.02916315, 0.24727069, 0.03179994,
       0.27353836, 0.06045644, 0.05103964, 0.05401903])
```

Combine the dataframe to know the shortlisted features

In [31]:

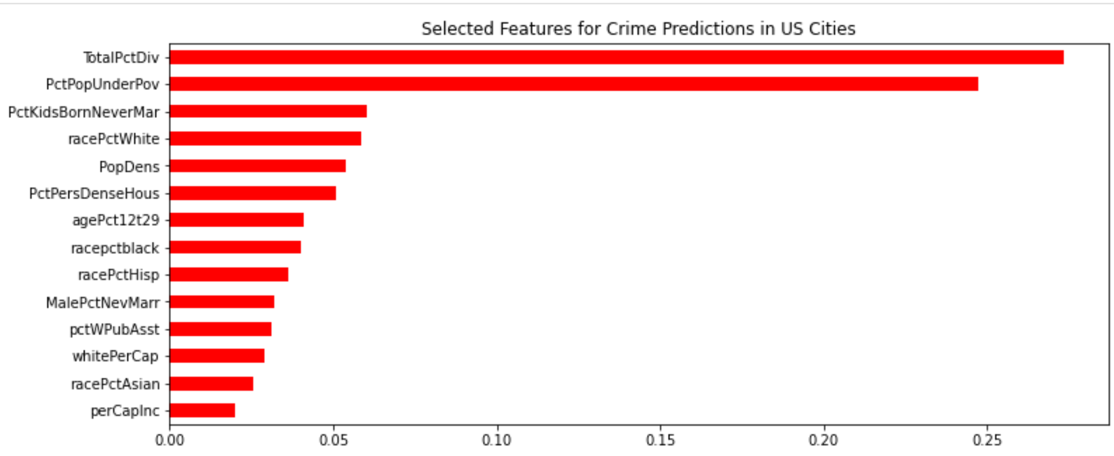
```
combination = pd.Series(importance,final_feature_cols)
combination
```

Out[31]:

```
racepctblack           0.040159
racePctWhite           0.058604
racePctAsian           0.025624
racePctHisp            0.036200
agePct12t29            0.041087
pctWPubAsst            0.030938
perCapInc              0.020102
whitePerCap            0.029163
PctPopUnderPov         0.247271
MalePctNevMarr         0.031800
TotalPctDiv            0.273538
PctKidsBornNeverMar    0.060456
PctPersDenseHous       0.051040
PopDens                0.054019
dtype: float64
```

In [32]:

```
combination.sort_values().plot.barh(color='red',figsize=(12,5))
plt.title("Selected Features for Crime Predictions in US Cities");
```



This model identified Total Percentage of People divorced as the number 1 predictor of crimes in the US communities. This was followed by the proportion of the population below poverty line. Again, the proportion of kids born to never married parents influences crime rates. Other factors include population density,percent of persons in dense housing (more than 1 person per room),proprotion of people between age 12 and 29 living in the neighborhood respectively.

Conclusion

There is a need for government to look into how to strengthen families becuase, it seems the major cause of crimes as identified from this exercise is a dysfunctional family.

In [ ]:

```

```