Data Mining for Pattern Discovery: How Income and Education Levels Affect Crime Rates.

Aashay Mokadam
Department of Computer Engineering
San José State University
San José, USA
aashay.mokadam@sjsu.edu

Mitwa Palkhiwala
Department of Computer Engineering
San José State University
San José, USA
mitwa.palkhiwala@sjsu.edu

Liubov Tovbin
Department of Computer Engineering
San José State University
San José, USA
liubov.tovbin@sjsu.edu

Rohit Sapkal
Department of Computer Engineering
San José State University
San José, USA
rohitbhagvat.sapkal@sjsu.edu

***Abstract*—In this work, we study the correlation between income, education, and crime rate in the US. We focus our attention on the publicly available data sets from the US Census Bureau and criminal records from the FBI from the years 2005-2009. An unexpected result is that certain education levels and income levels are negatively correlated. That is, higher education levels come along with the lower income levels in a given area. To predict crime rates, we test several machine learning algorithms: Linear Regression, Gradient Boosting Regressor, Decision Tree, K-neighbour Regressor, Random Forest Regressor, Ridge Regression, and Support Vector Regression (SVR). The SVR model performed the best, yielding the least mean squared error.**

***Index Terms*—crime prediction, regression, correlation**

I. Introduction

Crime is an age-old societal problem which impedes progress and adversely affects human societies. Factors that contribute to crime are far and many, and there is much value in being able to predict indicators of future criminal behavior amongst individuals. When we set aside the inevitable criminal personalities however, factors of low income, low intelligence, and a lack of empathy are often attributed to the development of a criminal lifestyle [1]. In this project, we look towards analysing mean income levels of households and average education levels, county-by-county, and their respective contributions to crime rates in the United States. We set out with the hypothesis that areas with a higher high-school dropout rate and lower per-capita incomes have a higher crime rate. Additionally, we analyse the relationships between violent crime and property crime, high-school dropout levels and rate of higher-education, and income distribution with respect to mean income levels.

II. Related Work

The problem of crime prevention interests researchers for a long time now. An abundance of publicly available data sets provides vast opportunities for the analysis. Various machine learning techniques have been tested on a subject of crime detection. S. Prabakaran et al. [5], for example, provide a comprehensive list of crime types along with the machine learning algorithms that best suit a given crime type detection. Probably, the best way to prevent a crime is to predict where and when it is going to happen. A significant amount of research from recent years concentrates on a problem of crime hotspot prediction. The researchers analyze demographic data of an area along with geospatial characteristics of the region in an attempt to predict whether or not this given area will be a crime hotspot. For example, S. Letourneau et al. [6] leverage the data about physical area characteristics, such as the presence of schools, libraries, parks, or liquor shops to understand the primary cause of crime in each neighborhood in Edmondo, Canada. S. Sathyadevan et al. [7] analyze crime records to detect patterns that lead to crime occurrences daily. Besides the relatively constant demographic and geospatial area characteristics, some researchers also look into features that change daily, or even hourly. H. Wang et al. [8] analyze taxi flow data. Jong-Min Kim et al. [9] consider climate conditions. B. Bogomolov et al. [14] study mobile phone aggregated data and derive second-order statistical features that describe demographic characteristics on an hourly basis. Moreover, B. Bogomolov et al. [14] ranked all the 6000 features available to them and discovered that the top 20 constitute the features of daily demographic statistics. The technical approaches also vary. To predict crime, the researchers use clustering and association rule mining [6], Naive Bayes, and Decision Tree algorithms [7], regression model [8]. B. Bogomolov et al. [14]] test 5 different models in the 5-fold cross-validation approach: Logistic Regression, Support Vector Machine, Neural Network, and Random Forest. All the mentioned research is mostly place-centric [16]- [12], meaning that the study concentrates on the area's topological characteristics. The place-centric approach is a relatively recent development [14]. Before that, researchers focused on a people-centric approach using demographic data to predict crime. A time-centric approach focuses on the temporal characteristics derived from the available crime records [7], [8], [14]. In our work, we would like to focus once again on a people-centric approach to analyze dependencies among crime, income, and education in the United States between the years 2005 to 2009. We investigate:

- Correlation between crime rates and income levels
- Correlation between crime rates and education levels
- Correlation between income and education

III. Data Used

A. *Data Collection*

The United States Census Bureau maintains historical records of demographic data that is very useful to us. For the purposes of our experiments, we have taken into consideration income and education data of individuals above the age of 25 years [3], in the time-span of the years 2005-2009. The "INC0X.xls" files contain the income data, and the "EDU0X.xls" provide us with education data. For crime levels, we refer to the UCR reports on crime statistics collected annually by the Federal Bureau of Investigation [4]; namely, the "Crime in the United States" sections. The latter is collected for the requisite years (2005-2009) and averaged in order to make the necessary comparisons uniform.

B. *Data Extraction and Preprocessing*

Education A large amount of fine-grained education data is available, with fields such as "Completed Less than 9th Grade", "High School Graduate", "Associate's Degree", and so on. From this, we extract the following fields for our evaluation:

- Percent of High School Dropouts
- Percent with No Degree (but have graduated High School)

We also define "Percent with Any Degree" for the purpose of convenient data analysis and inference. This is nothing more than the count of individuals not covered by the other two metrics.

Income Income data is available through both, average income measures (mean, median, per-capita), as well as frequency buckets. These buckets are as follows:

- less than $10,000
- $10,000 to $14,999
- $15,000 to $19,999
- $20,000 to $24,999
- $25,000 to $29,999
- $30,000 to $34,999
- $35,000 to $39,999
- $40,000 to $44,999
- $45,000 to $49,999
- $50,000 to $59,999
- $60,000 to $74,999
- $75,000 to $99,999
- $100,000 to $124,999
- $125,000 to $149,999
- $150,000 to $199,999
- $200,000 or more

We use the above to calculate the standard deviations of income, as well as to plot its general distribution trend. To accomplish this, we use the mid-point value of every income range. One challenge that is faced is that the "$200,000 or more" bucket has no exact value. Hence, in order to get an accurate measure of standard deviation, we calculate the "High Bucket Income" value for each county through the following:

$$I_{high\_bucket} = I_{mean} * F_{mean} - \sum_{s \epsilon S} I_s * F_s \qquad (1)$$

where I is the income for each bucket, F is the frequency, and S is the set of all buckets except "$200,000 or more". Finally, we analyze the correlation between all the income features in order to reduce the dimensionality of the problem, since these attributes are likely to be highly related to one another.

Unsurprisingly, fig 1 shows that the mean, median, and per-capita incomes are all strongly correlated. The income standard deviation does show some degree of variance, but ultimately follows suit as well. Now we are faced with the task of choosing between the mean and median income measures, since we can safely assume that the per-capita income is analogous to the mean household income.

Fig 2 plots the income frequency distribution throughout the entire country. A somewhat peculiar observation shows that the mean is higher than the median, but lower than the mode. In fact, the income buckets of `$60,000 to $74,999` and `$75,000 to $99,999` dominate the distribution. Due to this, the mean is skewed towards this range. However, since the `less than $10,000` bucket also has a larger-than-usual value, it makes the most sense to choose the median over the mean as a representative average income. As a result, we retain only the **Median Income** and **Income Standard Deviation** attributes.

Crime The FBI Crime Data provides us with attributes for different types of crime, as well as aggregated data for "Property Crime" and "Violent Crime". We also define a "Total Crime" measure as the sum of the former two, and analyze all three attributes.

As seen in figure 3, the crime attributes are all highly correlated, meaning that neither "Violent Crime" nor "Property Crime" contributes towards the overall crime rate disproportionately. While the violent crime rate is indeed not as strongly correlated with the total crime rate as the property crime rate (0.900 vs 0.975), the difference is fairly negligible. Hence, we shall retain only the **Total Crime Rate** attribute for our evaluation.

C. *Data Correlation*

Our final chosen attributes are:

- Education
  - Percent of High School Dropouts
  - Percent with No Degree
  - Percent with Any Degree (100% - Percent of High School Dropouts - Percent with No Degree)
- Income
  - Median Income per Household

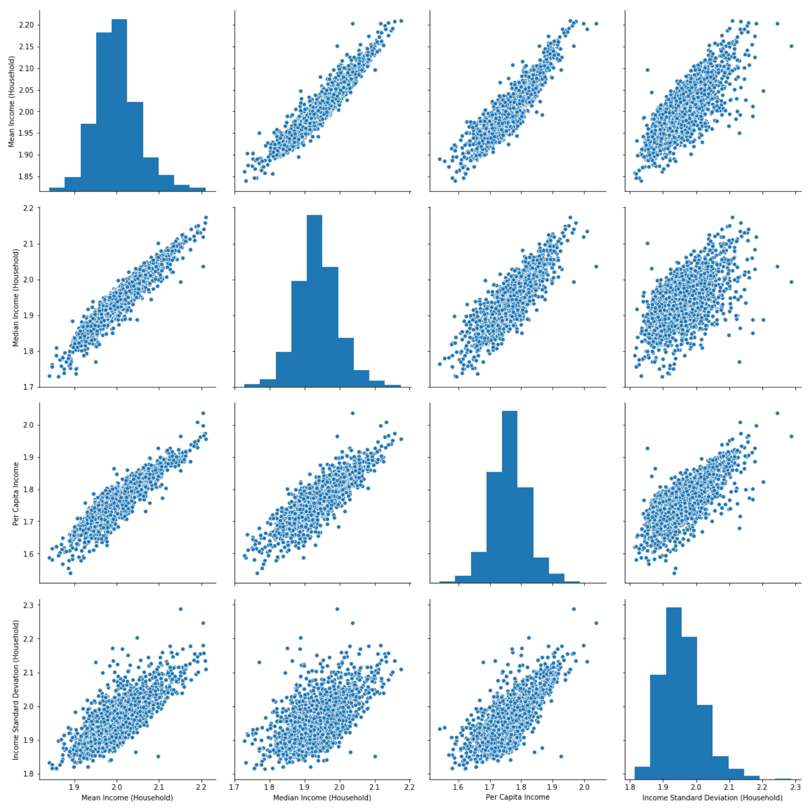

Fig. 1. Correlation between Mean Income, Median Income, Per Capita Income, and Income Standard Deviation

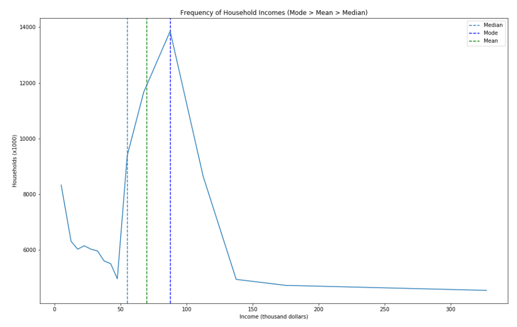

Fig. 2. Mean, Median, and Mode Incomes on the Income Distribution (Nationwide)

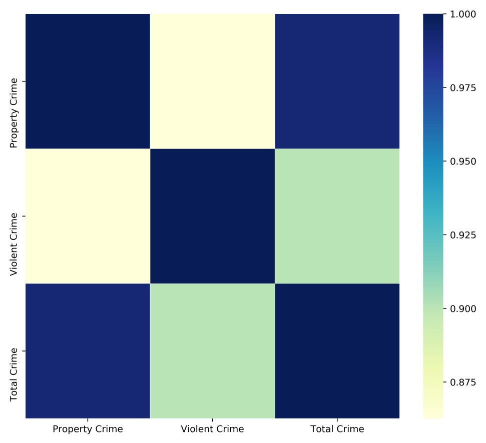

Fig. 3. Correlation between different types of crime

– Standard Deviation of Income

– Crime

– Total Crime Rate

Fig 4 shows the relation between the selected fields of income, education and crime. There are quite a few interesting points that can be inferred from this pair-plot diagram.

- The high school dropout rate is somewhat positively correlated with the percent of people with degrees. This could indicate that education levels tend to be symmetric across extremes of the distribution, i.e. in counties with high dropout rates, those who do not drop out of high school tend to pursue their education further.
- The high school dropout rate is negatively correlated with median income levels. This is intuitively expected, as people who do not finish high school are less likely to have lucrative careers.
- The percent of people with degrees (Associate's, Bache-

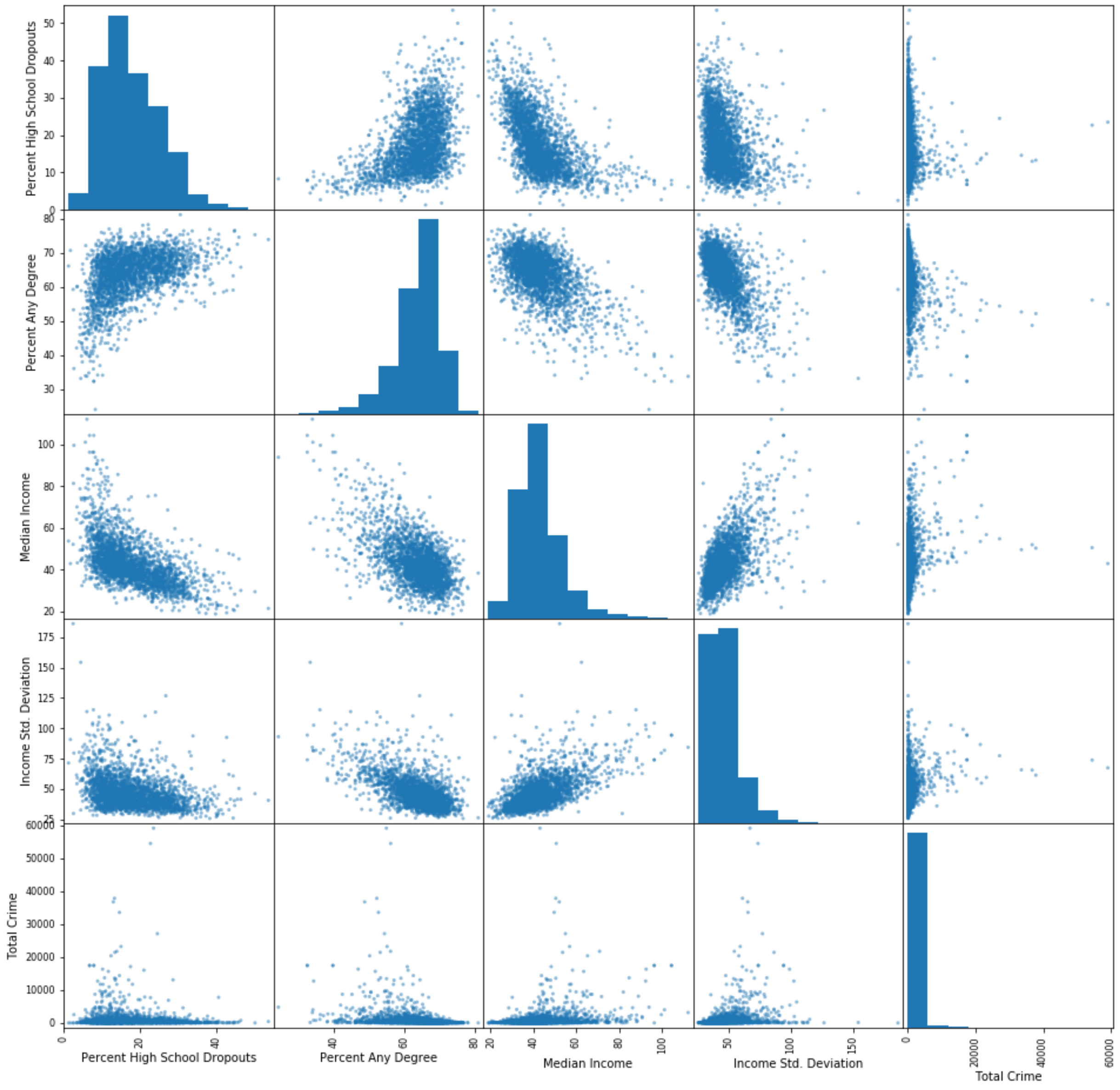

Fig. 4. Correlation between all features

lor's, and Graduate Degrees) is negatively correlated with median income levels. This is an unexpected insight, as one would assume that income levels increase in direct proportion to education levels. This could imply that people who pursue higher studies and delay their entry into the workforce do not see increases in their income significant enough to make up for lost time. Since our dataset inherently deals only with individuals above the age of 25, it is unlikely that students who are yet to graduate skew the data towards lower income levels.

- The aggregate effect of the above two points is that counties with higher rates of people graduating high

school and not pursuing further studies generally have higher median incomes.
- Finally, while income and education measures seem to have slightly different correlations with the total crime rate, at first glance, it does seem like crime is largely independent of the two factors.

In order to get an accurate numerical estimation of the contribution of education and income towards the crime rate, we implement the strategy described in Guyon et al [13], i.e. we fit a linear model and then examine the weights assigned to each of the features. A higher numerical value of weights indicates a higher contribution towards the crime rate.

IV. Prediction Models

To pick the best prediction model for the project, we applied a number of regression models on the features and label. The regression techniques are used to scrutinize two main factors: which particular features are significantly contributing to the outcome and if the features selected for the analysis are really playing a role in predicting the outcome. The regression models used in the projects are Linear Regression, Gradient Boosting Regressor, Decision Tree Regressor, K-nearest-neighbours Regressor, Random Forest Regressor, Ridge Regression and Support Vector Regression. We calculated the Mean Squared Error of each of the model and compared the results under various categories of education and income. The minimum mean squared error was obtained by Support Vector Regression, followed closely by Linear and Ridge Regressions. The models used are described as following:

A. Linear Regression

Linear Regression is a basic linear model generally used for predictive analysis. It focuses on finding the relationship between one dependent and one or more independent variables. It is useful in projects to evaluate various trends in the dataset, to forecast the future values or make estimations wherever needed. Suppose that we have n data pairs where (x_i, y_i), i = 1... n. The underlying relationship between y_i and x_i involving the error term c can be calculated using the formula: $y = a + b(x_i) + c$ [2].

B. Gradient Boosting Regressor

Gradient Boosting Regressor is an ensemble model. It is a model where the prediction is done sequentially. The current predictors learn from the mistakes of the previous predictors. The results having the highest errors are repeated most often for future predictions. So, the predictions are made based on errors rather than bootstrap process. Decision Trees, regressors, classifiers etc. can be taken into account for ensembling the gradient boosting regressor. As it learns from the past mistakes, it takes relatively lesser time for predicting the outcomes. However, the model can over-fit the data in some cases. Gradient descent can be used and by updating the predictions based on the rate of learning, minimum Mean Squared Error can be found out. [15]

C. Decision Tree Regressor

Decision Tree is a supervised learning model which is used for both regression and classification. It forms the models using a tree-like structure. It breaks down the dataset into smaller subsets until the last 2 leaf nodes are reached. The final tree has decision nodes and leaf nodes. The leaf nodes represent the decision (regression or classification). The topmost node is called the root node which is chosen after checking the best entropy or the most amount of information that it conveys. The root is usually considered as the best predictor. They can handle both types of data namely categorical and numerical. Some of the methods used in decision tree are ID3, C4.5, CART. The limitation of this model is that it may over-fit the data. Thus, to avoid over-fitting, the nodes with low information gain should be placed next to the high information gain nodes. [16]

D. K-nearest-neighbours Regressor

K nearest neighbors uses similarity measure to predict the numerical target values.One of the methods includes calculating the inverse weighted average of Euclidean distance, Manhattan distance, Minkowski distance for numerical values. For categorical values, the Hamming distance must be used to calculate the inverse weighted average. Another simple method is to calculate average of the numerical target of the K nearest neighbours. Generally, more the value of K, the more precise are the results as it reduces the overall noise. The ideal value for most of the datasets is observed to be 10 or more. It is a non-parametric technique and can be used in the pattern recognition and statistical estimations. [17]

E. Random Forest Regressor

Random Forest Regressor is a supervised learning method which uses ensemble regression trees for regression and classification. It is a bagging method in which various regression trees are implemented in a parallel fashion. The trees are independent of each other and can function without knowing the results of the other trees. There is no interaction between the trees while running. Each tree draws a random sample from the original dataset during the splitting process. This helps in reducing the chances of over-fitting. At the end, the outputs of all the trees are given votes and the output with the highest votes is chosen as final predictor for regression or classification. Random Forest is highly suitable for the problems having a large dataset. It gives high accuracy in finding the missing values in the dataset. [18]

F. Ridge Regression

Ridge Regression is a method for scrutinizing multiple regression data that shows a lot of collinearity in multiple feature. We get unbiased least squares estimates and large variances when such collinearity occurs. So, they may be far from the original value. By modifying the bias obtained by the regression techniques, ridge regression reduces the standard errors. Ridge regression can be considered similar to linear regression as both belongs to the linear model family. Thus, we even consider the assumptions of linearity, constant variance, and independence similar to the linear regression. [19]

G. Support Vector Regression

Support Vector Regression works on the same principle as Support Vector Machines. Given a training data set (x_1, y_1),... (x_n, y_n), the basic idea is to find a function f(x) that has at most c deviation from the target variable y_i for all the training data. This regression technique can work with the large datasets and even with the datasets having the missing values. As long as the deviation is fixed, only the support vectors at the deviation c are of importance. The other values does not contribute in the final outcome of the model. In other regression techniques, we focus on minimizing the error rate while in Support Vector Regression, we try to fit the errors within a certain threshold. [20]

V. Experiments, Challenges, and Results

A. Experimental Methodology

On our initial run, we got a large mean squared error using unscaled data. Following this, we tried a few transformations on the data to get better results. We observed a drastic change in the mean squared error and other parameters by making a few changes:

1) Scaling the income data to represent multiples of \$1000
2) Taking the log of the target variable.
3) Further using a standard scaler (from scikit-learn) yielded better performance

Playing around a bit more with the data, we noticed that the transformation of the target variable (crime rate) that worked the best was formed by normalizing the data and then applying the square root/4th root to $log1p(y)$ (where y is the target variable and the $log1p$ operation refers to $1+log$). The Table I shows the table representing the mean squared error calculated in every model used along with the education level and the mean or standard deviation income. After analyzing all the values of the mean squared error in the table, we came to a conclusion that the Support Vector Regression gave the least mean squared error in all the categories of education and income, followed by Linear Regression and Ridge Regression. Thus, we finalized the support vector regressor and linear regressor models for our project.

B. Discussion of Results

SVR # of Support Vectors = 745. This is a large number, and usually indicates over-fitting. On the other hand, this model obtained the lowest mean squared error. This furthers the hypothesis that crime rates are not as dependent on education and income levels as one would think.

Linear Regression The weights assigned by the Linear Regressor to each of the features are as follows: Education (High School Dropout Rate) - `0.02535872` Income (Median Income) - `0.04726192` The low magnitudes of the weights lead us to conclude that crime rates (by county) are not heavily reliant on education and income levels considered in this paper. Further, the relative magnitudes indicate that while crime rates are positively correlated with high school dropout rates, they are more positively correlated with median income levels. This does seem contrary to the hypothesis that low-income areas are more susceptible to crime.

Correlation between Crime and Higher Studies One

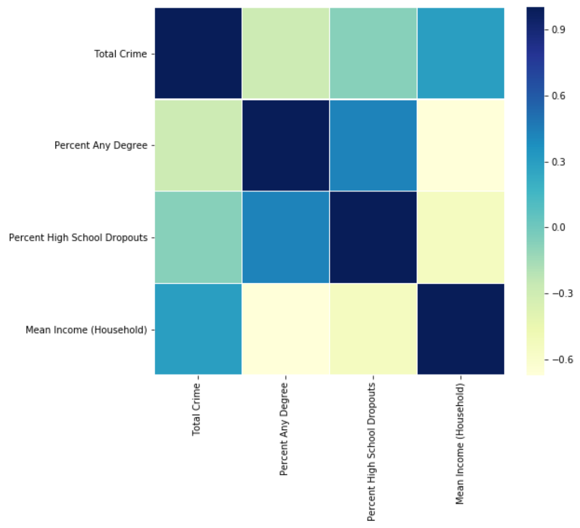

Fig. 5. Correlation between all features

hypothesis that is supported by out experiments is that the higher the percentage of people with degrees, the lower the crime rate. While this relationship isn't very clear in Fig 4, the heat-map in Fig 5 demonstrates the negative correlation between these two measures. This is further confirmed by running a linear regression with "Percent with Degrees" and "Median Income Rate" as features. The weights obtained are: Education (Rate of People with Degrees) - `-0.02912309` Income (Median Income) - `0.01654517` Compared to the previous experiment with the high school dropout rate in focus, this time, it is the education feature that has the higher magnitude, and the negative sign demonstrates a negative correlation.

VI. Further Work and Conclusion

Crime is ultimately a phenomenon with complex, compound factors that contribute to it. Through our experimentation, we have found that while county-wide census data does yield some interesting insights into education, income, and crime levels, it is not enough to simply consider them as features if we want to generate any sort of predictive model that captures the relation between them.

Our results show a positive correlation, albeit a weak one, between both high school dropout rates and income levels with respect to crime, and a negative correlation between rates of higher studies and crime. However, let us not neglect the fact that the rate of higher studies are also negatively correlated with income levels, which seem to be positively correlated with crime throughout. This could indicate that the positive correlation between income levels and crime could be coincidental.

TABLE I
RESULTS OF EXPERIMENTATION WITH DIFFERENT FEATURE CONFIGURATIONS

BoostingMSE	DecisionTreeMSE	KNeighbourMSE	LinearRegMSE	RandomForestMSE	RidgeMSE	SVR_MSE	crime_regularization	education_type	income_type
0.135687	0.142844	0.187286	0.135341	0.135687	0.135341	0.133981	2nd Root of Log	% High School Dropouts	Median
0.141375	0.150008	0.196002	0.145034	0.141375	0.145033	0.140423			Standard Deviation
0.139161	0.141787	0.188993	0.134803	0.139161	0.134802	0.142382		% Any Degree	Median
0.138185	0.140847	0.189016	0.137201	0.138185	0.137202	0.136058			Standard Deviation
0.020676	0.022114	0.029812	**0.020799**	0.020676	**0.020799**	**0.020618**	**4th Root of Log**	**% High School Dropouts**	**Median**
0.021839	0.021925	0.030625	0.022015	0.021839	0.022014	0.021241			Standard Deviation
0.021476	0.021721	0.029206	0.020920	0.021476	0.020920	0.021868		% Any Degree	Median
0.021439	0.021700	0.030082	0.021203	0.021439	0.021203	0.021044			Standard Deviation

One must also consider that the income and education levels are correlated, but not directly. Instead, the distribution of average education levels into the buckets of "High School Dropout" and "High School Graduate Without a Degree" plays a part in determining the median education level. Another point to consider is that the income levels are heavily skewed towards the `$60,000 to $99,999` range. Further work that accounts for the correlation between income and education levels, as well as their own internal distributions could yield more conclusive results towards prediction of crime rates.

REFERENCES

[1] Farrington, David P., and Brandon C. Welsh. Saving children from a life of crime: Early risk factors and effective interventions. Oxford University Press, 2008.

[2] Statistics Solutions. (2019). What is Linear Regression?. [online] Available at: https://www.statisticssolutions.com/what-is-linear-regression/ [Accessed 26 Nov. 2019].

[3] https://www.census.gov/library/publications/2011/compendia/usa-counties-2011.html

[4] https://www.fbi.gov/services/cjis/ucr

[5] S. Prabakaran, S. Mitra, "Survey of Analysis of Crime Detection Techniques Using Data Mining and Machine Learning," National Conference on Mathematical Techniques and its Applications (NCMTA 18) IOP Publishing IOP Conf. Series: Journal of Physics: Conf. Series 1000 (2018) 012046, DOI 10.1088/1742-6596/1000/1/012046

[6] Letourneau, Steven, et al. "The effects of neighbourhood characteristics on crime incidence," 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2018. DOI: 10.1109/CCWC.2018.8301675

[7] Sathyadevan, Shiju, et al. "Crime Analysis and Prediction Using Data Mining ." 2014 First International Conference on Networks & Soft Computing (ICNSC2014). IEEE, 2014. DOI: 10.1109/CNSC.2014.6906719

[8] Wang, Hongjian, et al. "Crime Rate Inference with Big Data." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16. Vol. 13, ACM Press, 2016. DOI: 10.1145/2939672.2939736

[9] Jong-Min Kim, Hwang-Kwon Ahn and Dong-Hwi Lee, "A study on the occurrence of crimes due to climate changes using decision tree." IT Convergence and Security 2012. 1st Ed. 2013. ed. Vol. 215. IT Convergence and Security 2012, Dordrecht, Springer, Netherlands, 2013. DOI:10.1007/978-94-007-5860-5-125

[10] C.H. Yu, M. Ward, M. Morabito, and W. Ding, "Crime Forecasting Using Data Mining Techniques." 2011 IEEE 11th International Conference on Data Mining Workshops. DOI: 10.1109/ICDMW.2011.56

[11] Lee, Ickjai, and Estivill-Castro, Vladimir. "Exploration of massive crime datasets through data mining techniques." Applied Artificial Intelligence 25, no. 5 (2011): 362-79. DOI: 10.1080/08839514.2011.570153

[12] Phillips, Peter, and Lee, Ickjai. "Mining Co-distribution Patterns for Large Crime Datasets." Expert Systems with Applications 39, no.14 (2012): 11556-1563. DOI: 10.1016/j.eswa.2012.03.071

[13] Guyon, I., Weston, J., Barnhill, S. et al. Machine Learning (2002) 46: 389. https://doi.org/10.1023/A:1012487302797

[14] Bogomolov, B. Lepri, J. Staiano, N. Oliver, F. Pianesi, A. Pentland, "Once Upon a Crime: Towards Crime Prediction from Demographics and Mobile Data," ICMI '14 Proceedings of the 16th International Conference on Multimodal Interaction, pp 427-434, 2014, DOI 10.1145/2663204.2663254

[15] Medium. (2019). Understanding Gradient Boosting Machines. [online] Available at: https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab [Accessed 26 Nov. 2019].

[16] Medium. (2019). Decision Trees Explained Easily. [online] Available at: https://medium.com/@chiragsehra42/decision-trees-explained-easily-28f23241248 [Accessed 26 Nov. 2019].

[17] Saedsayad.com. (2019). KNN Regression. [online] Available at: "https://www.saedsayad.com/knearestneighborsreg.html [Accessed 26 Nov. 2019].

[18] Medium. (2019). Random Forest and its Implementation. [online] Available at: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f [Accessed 26 Nov. 2019].

[19] Ncss-wpengine.netdna-ssl.com. (2019). [online] Available at: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge-Regression.pdf [Accessed 26 Nov. 2019].

[20] Medium. (2019). Support Vector Regression Or SVR. [online] Available at: https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff [Accessed 26 Nov. 2019].

VII. APPENDIX

Listing 1. Data Extraction and Preprocessing

```
import os
try:
        os.chdir(os.path.join(os.getcwd(), '
            ↪ code'))
        print(os.getcwd())
except:
        pass
# %%
import pandas as pd
import numpy as np
import sys
import os
import re

import us_state_abbrev as states

base_path = os.path.abspath("..")

# %% [markdown]
# ## Crime

# %%
crime_path = os.path.join(base_path, 'data/
    ↪ crime/')
crime_dataframes = [os.path.join(crime_path,
    ↪ file) for file in os.listdir(crime_path)
    ↪ ]

def GetCrimeCountyNames(row):
    state = row.name[0]
    state = state.split("\ue83a")[0]
    state = state.split("-")[0]
    state = state.lower().title().strip()
    state_code = states.us_state_abbrev[state]
    county = row.name[1]
    county = "".join([letter for letter in
        ↪ county if not letter.isnumeric()])
    county = re.sub("County" ,"", county)
    county = re.sub("Police" ,"", county)
    county = re.sub("Department" ,"", county)
    county = county.strip()

    full_county = county + ", " + state_code.
        ↪ strip()

    return full_county

def GetCrimeDataFrame(files):

    dataframes = []

    for file in files:
        df = pd.read_excel(file, index_col
            ↪ =[0,1], index_row=0, encoding='
            ↪ utf-8')
        df.columns = df.columns.str.lower()
        df["Area_name"] = df.apply(lambda row:
            ↪  GetCrimeCountyNames(row), axis
            ↪ =1)
        df = df.set_index("Area_name")
        df["total crime"] = df.apply(lambda
            ↪ row: row["violent crime"] + row[
            ↪ "property crime"], axis=1)
        df["log violent crime"] = df.apply(
            ↪ lambda row: np.log1p(row["
            ↪ violent crime"]) if row["violent
            ↪  crime"] != 0 else 0, axis=1)
        df["log property crime"] = df.apply(
            ↪ lambda row: np.log1p(row["
            ↪ property crime"]) if row["
            ↪ property crime"] != 0 else 0,
            ↪ axis=1)
        df["log total crime"] = df.apply(
            ↪ lambda row: np.log1p(row["total
            ↪ crime"]) if row["total crime"]
            ↪ != 0 else 0, axis=1)
        df["root log violent crime"] = df.
            ↪ apply(lambda row: np.power(row["
            ↪ log violent crime"], 1/2), axis
            ↪ =1)
        df["root log property crime"] = df.
            ↪ apply(lambda row: np.power(row["
            ↪ log property crime"], 1/2), axis
            ↪ =1)
        df["root log total crime"] = df.apply(
            ↪ lambda row: np.power(row["log
            ↪ total crime"], 1/2), axis=1)
        df.head()
        dataframes.append(df)

    final_columns = list(set.intersection(*map
        ↪ (set, [dframe.columns.tolist() for
        ↪ dframe in dataframes])))
    crime_data = pd.concat(dataframes, sort=
        ↪ False)

    by_row_index = crime_data.groupby(
        ↪ crime_data.index)
    crime_data = by_row_index.mean()
    crime_data = crime_data[final_columns]
    final_columns = [col.title() for col in
        ↪ final_columns]
    crime_data.columns = final_columns

    crime_data = crime_data.fillna(0)
    crime_data = crime_data.loc[crime_data["
        ↪ Total Crime"] != 0.0]

    return crime_data

def LoadCrimeData():

    crime_data = GetCrimeDataFrame(
        ↪ crime_dataframes)
    cols_to_return = ["Property Crime", "
        ↪ Violent Crime", "Total Crime", "Log
        ↪ Property Crime", "Log Violent Crime"
        ↪ , "Log Total Crime", "Root Log
        ↪ Property Crime", "Root Log Violent
        ↪ Crime", "Root Log Total Crime"]
    return crime_data[cols_to_return]

def LoadViolentCrimeData():
    crime_data = GetCrimeDataFrame(
        ↪ crime_dataframes)
    cols_to_return = ["Murder And Nonnegligent
        ↪  Manslaughter", "Forcible Rape", "
        ↪ Robbery", "Aggravated Assault"]
    return crime_data[cols_to_return]

def LoadPropertyCrimeData():
    crime_data = GetCrimeDataFrame(
        ↪ crime_dataframes)
    cols_to_return = ["Burglary", "Larceny-
        ↪ Theft", "Motor Vehicle Theft", "
        ↪ Arson1"]
    return crime_data[cols_to_return]

# %% [markdown]
# ## Education

# %%
education_path = os.path.join(base_path, 'data
    ↪ /education/')

def GetNoDegree(row):

    total = int(row["Total"])
    no_degree = int(row["Completing less than
        ↪ 9th grade"]) + int(row["High school
        ↪ graduate (includes equivalency)"]) +
        ↪  int(row["Some college, no degree"])
    return ((1.00 - (float(no_degree)/float(
        ↪ total)))*100)


def CleanEducationAttrName(row):
    name = row["Attribute Name"]
    name = re.sub(r"Educational attainment", "
        ↪ ", name)
    name = re.sub(r"Persons 25 years and over,
        ↪ ", "", name)
    name = re.sub(r"persons 25 years and over"
        ↪ , "", name)
    name = re.sub(r"2005-2009", "", name)
    name = re.sub(r"\s\s+", " ", name)
    name = re.sub("\-", "", name)
    name = name.strip().capitalize()
    return name

def GetEducationDataFrame(education_path):
    education_dataframes = [os.path.join(
        ↪ education_path, file) for file in os
        ↪ .listdir(education_path) if "EDU" in
        ↪  file]

    education_metadata = pd.read_excel(os.path
        ↪ .join(education_path, "
        ↪ education_by_counties.xlsx"))
    education_metadata["Attribute Name"] =
        ↪ education_metadata.apply(lambda row:
        ↪  CleanEducationAttrName(row), axis
        ↪ =1)

    education_data = []

    for idx, row in education_metadata.
        ↪ iterrows():
        location = row["Location"]
        filename = location[:len(location)-1]
        file = [x for x in
            ↪ education_dataframes if re.
            ↪ search(filename, x)][0]
        df = pd.read_excel(file, location,
            ↪ index_col=0)
        column = df.loc[:,[row["ID"]]]
        column = column.rename(columns = {row[
            ↪ 'ID']:row["Attribute Name"]})
        education_data.append(column)

    education_data = pd.concat(education_data,
        ↪  axis=1)
    education_data = education_data.loc[
        ↪ education_data["Total"] != 0]

    return education_data


def LoadEducationData():

    education_data = GetEducationDataFrame(
        ↪ education_path)
    education_data["Percent High School
        ↪ Dropouts"] = education_data.apply(
        ↪ lambda row: 100.00 - float(row["
        ↪ Percent high school graduate or
        ↪ higher"]), axis=1)
    education_data["Percent No Degree"] =
        ↪ education_data.apply(lambda row:
        ↪ GetNoDegree(row), axis=1)
    education_data["Percent Any Degree"] =
        ↪ education_data.apply(lambda row:
        ↪ 100.00 - float(row["Percent No
        ↪ Degree"]), axis=1)

    return education_data[["Percent High
        ↪ School Dropouts", "Percent No Degree
        ↪ ", "Percent Any Degree"]]

# %% [markdown]
# ## Income

# %%
income_path = os.path.join(base_path, 'data/
    ↪ income/')

def GetIncomeRangeMeans():
    income_range_means = {
        "Households with income less than \$10
            ↪ ,000": 5000,
        "Households with income of \$10,000 to
            ↪  \$14,999": 12500,
        "Households with income of \$15,000 to
            ↪  \$19,999": 17500,
        "Households with income of \$20,000 to
            ↪  \$24,999": 22500,
        "Households with income of \$25,000 to
            ↪  \$29,999": 27500,
        "Households with income of \$30,000 to
            ↪  \$34,999": 32500,
        "Households with income of \$35,000 to
            ↪  \$39,999": 37500,
        "Households with income of \$40,000 to
            ↪  \$44,999": 42500,
        "Households with income of \$45,000 to
            ↪  \$49,999": 47500,
        "Households with income of \$50,000 to
            ↪  \$59,999": 55000,
        "Households with income of \$60,000 to
            ↪  \$74,999": 67500,
        "Households with income of \$75,000 to
            ↪  \$99,999": 87500,
        "Households with income of \$100,000
            ↪ to \$124,999": 112500,
        "Households with income of \$125,000
            ↪ to \$149,999": 137500,
        "Households with income of \$150,000
            ↪ to \$199,999": 175000
    }
    return income_range_means

income_range_means = GetIncomeRangeMeans()

def GetIncomeStdDeviation(row):

    values, frequencies = [], []

    total_income = int(row["Households with
        ↪ income, total"]) * int(row["Mean
        ↪ Income (Household)"])
    total_high_bracket = int(row["Households
        ↪ with income of \$200,000 or more"])
    columns_to_check = [str(x) for x in row.
        ↪ axes[0].tolist() if "$" in str(x)
        ↪ and "more" not in str(x)]

    total_without = 0
    for c in columns_to_check:
        values.append(float(income_range_means
            ↪ [c])/1000.0)
        frequencies.append(int(row[c]))
        total_without += int(row[c]) * float(
            ↪ income_range_means[c]/1000.0)

    remainder = total_income - total_without
    high_bracket_mean = 0

    if total_high_bracket != 0:
        high_bracket_mean = float(remainder/
            ↪ total_high_bracket)
    values.append(int(high_bracket_mean))
    frequencies.append(int(total_high_bracket)
        ↪ )

    overall_income_data = np.repeat(np.array(
        ↪ values), np.array(frequencies))
    std_dev = np.std(overall_income_data,
        ↪ dtype=np.float64, ddof=1)

    return std_dev, high_bracket_mean


def CleanIncomeAttrName(row):
    name = row["Attribute Name"]
    name = re.sub(r" in the past 12 months \(
        ↪ in 2009 inflation-adjusted dollars\)
        ↪ ", "", name)
    name = re.sub(r"in 2005-2009", "", name)
    name = re.sub(r"2005-2009", "", name)
    name = re.sub(r"\s\s+", " ", name)
    name = re.sub("\$", "\\$", name)
    name = name.strip()
    return name


def GetIncomeDataFrame(income_path):

    income_dataframes = [os.path.join(
        ↪ income_path, file) for file in os.
        ↪ listdir(income_path) if "INC" in
        ↪ file]

    income_metadata = pd.read_excel(os.path.
        ↪ join(income_path, "
        ↪ income_by_counties.xlsx"))
    income_metadata["Attribute Name"] =
        ↪ income_metadata.apply(lambda row:
        ↪ CleanIncomeAttrName(row), axis=1)

    income_data = []

    for idx, row in income_metadata.iterrows()
        ↪ :
        location = row["Location"]
        filename = location[:len(location)-1]
        file = [x for x in income_dataframes
            ↪ if re.search(filename, x)][0]
        df = pd.read_excel(file, location,
            ↪ index_col=0)
        column = df.loc[:,[row["ID"]]]
        column = column.rename(columns = {row[
            ↪ 'ID']:row["Attribute Name"]})
        income_data.append(column)

    income_data = pd.concat(income_data, axis
        ↪ =1)
    income_data = income_data.rename(columns =
                    {"Mean household income":
                        ↪ "Mean Income (
                        ↪ Household)",
                    "Median household income":
                        ↪  "Median Income (
                        ↪ Household)",
                    "Per capita income": "Per
                        ↪ Capita Income"})
    income_data["Mean Income (Household)"] =
        ↪ income_data["Mean Income (Household)
        ↪ "].astype(float)/1000.00
    income_data["Median Income (Household)"] =
        ↪  income_data["Median Income (
        ↪ Household)"].astype(float)/1000.00
    income_data["Per Capita Income"] =
        ↪ income_data["Per Capita Income"].
        ↪ astype(float)/1000.00

    income_data = income_data.loc[income_data[
        ↪ "Mean Income (Household)"] != 0.0]
    income_data["Income Standard Deviation (
        ↪ Household)"], income_data["High
        ↪ Bracket Income (Household)"] = zip(*
        ↪ income_data.apply(lambda row:
        ↪ GetIncomeStdDeviation(row), axis=1))

    return income_data


def LoadIncomeData():

    income_data = GetIncomeDataFrame(
        ↪ income_path)
    return income_data[["Mean Income (
        ↪ Household)", "Median Income (
        ↪ Household)",
                    "Per Capita Income", "
                        ↪ Income Standard
                        ↪ Deviation (
                        ↪ Household)", "
                        ↪ High Bracket
                        ↪ Income (
                        ↪ Household)"]]


def LoadIncomeDistribution():

    income_data = GetIncomeDataFrame(
        ↪ income_path)
    income_range_means = GetIncomeRangeMeans()
    income_range_means["Households with income
        ↪  of \$200,000 or more"] =
        ↪ income_data.loc["UNITED STATES", "
        ↪ High Bracket Income (Household)"
        ↪ ]*1000.00

    columns_to_drop = ["Median Income (
        ↪ Household)", "Mean Income (Household
        ↪ )",
                "Households with income, total
                    ↪ ", "Per Capita Income",
                "Income Standard Deviation (
                    ↪ Household)", "High
                    ↪ Bracket Income (
                    ↪ Household)"]

    income_data = income_data.drop(
        ↪ columns_to_drop, axis=1).loc["UNITED
        ↪  STATES"]

    final_data = income_data.rename(lambda x:
        ↪ str(float(income_range_means[x])
        ↪ /1000))

    return final_data
```

```
# %% [markdown]
# ## Processed Data

# %%
def get_processed_data():

    file_path = os.path.join(base_path, '
        ↪ preprocessed/education_income_crime.
        ↪ xlsx')

    if os.path.exists(file_path):
        master_df = pd.read_excel(file_path,
            ↪ index_col=0)

    else:
        crime_data = LoadCrimeData()
        education_data = LoadEducationData()
        income_data = LoadIncomeData()

        master_df = crime_data.merge(
            ↪ income_data, on="Area_name").
            ↪ merge(education_data, on="
            ↪ Area_name")
        master_df.to_excel(file_path)

    return master_df
```

Listing 2. Helper to Access Data

```
#!/usr/bin/env python
# coding: utf-8

# In[6]:


from preprocess import get_processed_data,
    ↪ LoadViolentCrimeData,
    ↪ LoadPropertyCrimeData,
    ↪ LoadIncomeDistribution,
    ↪ LoadEducationData
data = get_processed_data()


# In[7]:


def get_label(crime_type = "total", reg = "
    ↪ sqrt_log"):
    """
    Returns a county-indexed dataframe of
        ↪ labels(y) based on crime_type
    :param crime_type: {"property", "violent",
        ↪  "total"}
    :param reg: {"none", "log", "sqrt_log"}
    :return: pandas dataframe
    """

    crime = ""
    if reg == "log":
        crime += "Log "
    elif reg == "sqrt_log":
        crime += "Root Log "

    if crime_type == "property":
        crime += "Property Crime"
    elif crime_type == "violent":
        crime += "Violent Crime"
    else:
        crime += "Total Crime"

    return data[[crime]]


def get_features(education_type="dropout",
↪ income_type="mean", get_high_bracket =
↪ False):
    """
    Returns a county-indexed dataframe of
        ↪ features(X = [education, income])
        ↪ based on params
    :param education_type: {"dropout", "
        ↪ degreeless", "degree"}
    :param income_type: {"mean", "median", "
        ↪ percapita", "deviation"}
    :param get_high_bracket: {True, False}
    :return: pandas dataframe
    """

    feature_columns = []

    if education_type == "degreeless":
        feature_columns.append("Percent No
            ↪ Degree")
    elif education_type == "dropout":
        feature_columns.append("Percent High
            ↪ School Dropouts")
    else:
        feature_columns.append("Percent Any
            ↪ Degree")

    if income_type == "median":
        feature_columns.append("Median Income
            ↪ (Household)")
    elif income_type == "percapita":
        feature_columns.append("Per Capita
            ↪ Income")
    elif income_type == "deviation":
        feature_columns.append("Income
            ↪ Standard Deviation (Household)")
    else:
        feature_columns.append("Mean Income (
            ↪ Household)")

    if get_high_bracket == True:
        feature_columns.append("High Bracket
            ↪ Income (Household)")

    return data[feature_columns]


def get_data(education_type="dropout",
↪ income_type="mean", crime_type = "total"
↪ , crime_reg = "sqrt_log",
↪ get_high_bracket = True):
    """
    Returns a county-indexed dataframe of
        ↪ features (education, income) and
        ↪ label (crime) based on params
    :param education_type: {"dropout", "
        ↪ degreeless", "degree"}
    :param income_type: {"mean", "median", "
        ↪ percapita", "deviation"}
    :param crime_type: {"property", "violent",
        ↪  "total"}
    :param crime_reg: {"none", "log", "
        ↪ sqrt_log"}
    :return: pandas dataframe
    """

    label = get_label(crime_type, crime_reg)
    features = get_features(education_type,
        ↪ income_type, get_high_bracket)

    queried_data = features.merge(label, on="
        ↪ Area_name")

    return queried_data


def get_all_data():
    """
    Returns a county-indexed dataframe of all
        ↪ feature types (X) and label types (y
        ↪ )
    Refer to schema or documentation of the
        ↪ get_data function for information
        ↪ about columns
    Preferably use for exploration and
        ↪ visualization only.
    Use the get_data, get_features, or
        ↪ get_label functions to retrieve data
        ↪  for processing
    :return: pandas dataframe
    """

    return data


def get_us_income_distribution():
    """
    Returns the income distribution for all
        ↪ income ranges in the United States
    :return: pandas series
    """
    return LoadIncomeDistribution()

def get_all_property_crime():
    """
    Returns all the property crime data
    :return: pandas dataframe
    """
    return LoadPropertyCrimeData()

def get_all_violent_crime():
    """
    Returns all the violent crime data
    :return: pandas dataframe
    """
    return LoadViolentCrimeData()

def get_all_education_data():
    """
    Returns all education data (states
        ↪ included)
    :return: pandas dataframe
    """
    return LoadEducationData()
```

```
# In[ ]:
```

Listing 3. Model Training, Evaluation

```
import os
try:
        os.chdir(os.path.join(os.getcwd(), '
            ↪ code'))
        print(os.getcwd())
except:
        pass
# %%
from IPython import get_ipython

# %%
import reader
import numpy as np
import pandas as pd
from sklearn.linear_model import Ridge,
    ↪ LinearRegression, LogisticRegression
from sklearn.model_selection import
    ↪ train_test_split
import sklearn.metrics as metrics
from matplotlib import pyplot as plt
get_ipython().run_line_magic('matplotlib', '
    ↪ inline')
import seaborn as sns

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import
    ↪ RandomForestRegressor,
    ↪ GradientBoostingRegressor
from sklearn.neighbors import
    ↪ KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import
    ↪ StandardScaler
from sklearn.model_selection import
    ↪ ParameterGrid

# %% [markdown]
# #### Loading the dataset

# %%
data=reader.get_all_data()
data.head()

# %% [markdown]
# #### Getting features and label and
    ↪ performing few transformations

# %%
from collections import OrderedDict
class Row(object):

    def __init__(self):
        self.education_type = None
        self.income_type = None
        self.crime_regularization = None
        self.LinearRegMSE = None
        self.RidgeMSE = None
        self.DecisionTreeMSE = None
        self.KNeighbourMSE = None
        self.SVR_MSE = None
        self.RandomForestMSE = None
        self.BoostingMSE = None
    def toDict(self):
        return {'education_type' : self.
            ↪ education_type,
                'income_type' : self.
                    ↪ income_type,
                'crime_regularization': self.
                    ↪ crime_regularization,
                'LinearRegMSE':  self.
                    ↪ LinearRegMSE,
                'RidgeMSE' : self.RidgeMSE,
                'DecisionTreeMSE' : self.
                    ↪ DecisionTreeMSE,
                'KNeighbourMSE': self.
                    ↪ KNeighbourMSE,
                'SVR_MSE' : self.SVR_MSE,
                'RandomForestMSE': self.
                    ↪ RandomForestMSE,
                'BoostingMSE' :self.
                    ↪ BoostingMSE
               }

# %% [markdown]
# #### Applying models

# %%
import warnings
warnings.filterwarnings("ignore")

param_grid = {"education_type" :["dropout", "
    ↪ degree"], "income_type" :["median", "
    ↪ deviation"], "crime_type":["log", "
    ↪ sqrt_log"]}
result = pd.DataFrame()

for param in list(ParameterGrid(param_grid)):
    row = Row()
    row.education_type = "High School Dropout
        ↪ Percent" if param['education_type']
        ↪ == 'dropout' else "Percent with Any
        ↪ Degree"
    row.income_type = "Median Income" if param
        ↪ ['income_type'] == "median" else "
        ↪ Income Standard Deviation"
    row.crime_regularization = "Square Root of
        ↪  Log" if param['crime_type'] == 'log
        ↪ ' else "Fourth Root of Log"
    X = reader.get_features(param['
        ↪ education_type'], param['income_type
        ↪ '])
    y = reader.get_label('total', param['
        ↪ crime_type'])
    y=np.power(y,1/2)
    X_train, X_test, y_train, y_test =
        ↪ train_test_split(X, y, test_size
        ↪ =0.2, random_state=0)


    ## Scale input data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train.
        ↪ to_numpy())
    X_test = scaler.fit_transform(X_test.
        ↪ to_numpy())

    ## Liner Regression
    linreg = LinearRegression()
    linreg.fit(X_train, y_train)
    y_pred=linreg.predict(X_test)
    row.LinearRegMSE = metrics.
        ↪ mean_squared_error(y_test, y_pred)

    ## Ridge Regression
    ridgereg = Ridge(alpha=1.0)
    ridgereg=ridgereg.fit(X_train,y_train)
    y_pred=ridgereg.predict(X_test)
    row.RidgeMSE = metrics.mean_squared_error(
        ↪ y_test, y_pred)

    ## Decision Tree
    regr = DecisionTreeRegressor(max_depth=2)
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    row.DecisionTreeMSE = metrics.
        ↪ mean_squared_error(y_test, y_pred)

    ## Random Forest Tree
    regr = RandomForestRegressor(max_depth=2)
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    row.RandomForestMSE = metrics.
        ↪ mean_squared_error(y_test, y_pred)

    ### Boosting
    params = {'n_estimators': 100, 'max_depth'
        ↪ : 2}
    clf = GradientBoostingRegressor(**params)
    clf.fit(X_train, y_train)
    row.BoostingMSE = metrics.
        ↪ mean_squared_error(y_test, y_pred)

    ### KNN
    neigh = KNeighborsRegressor(n_neighbors=3)
    neigh.fit(X_train, y_train)
    y_pred=neigh.predict(X_test)
    row.KNeighbourMSE = metrics.
        ↪ mean_squared_error(y_test, y_pred)

    ### SVR
    svr = SVR(gamma='auto')
    svr = svr.fit(X_train, y_train.values.
        ↪ ravel())
    y_pred=svr.predict(X_test)
    row.SVR_MSE = metrics.mean_squared_error(
        ↪ y_test, y_pred)


    result = result.append(row.toDict(),
        ↪ ignore_index=True)

result

# %% [markdown]
# #### SVR Support Vectors

# %%
X = reader.get_features("dropout", "median")
y = reader.get_label('total', "sqrt_log")
y=np.power(y,1/2)
X_train, X_test, y_train, y_test =
    ↪ train_test_split(X, y, test_size=0.2,
    ↪ random_state=0)

scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train.
    ↪ to_numpy())
X_test = scaler.fit_transform(X_test.to_numpy
    ↪ ())


svr = SVR(gamma='auto')
svr = svr.fit(X_train, y_train.values.ravel())
y_pred=svr.predict(X_test)

print(metrics.mean_squared_error(y_test,
    ↪ y_pred))
supportVectors = len(svr.support_vectors_)
SV=svr.support_vectors_
print("Number of support vectors : ",
    ↪ supportVectors)
print(" ")
print("Support Vectors : \n", SV)

# %% [markdown]
# #### Linear Regression

# %%
X = reader.get_features("dropout", "median")
y = reader.get_label('total', "sqrt_log")
y=np.power(y,1/2)
X_train, X_test, y_train, y_test =
    ↪ train_test_split(X, y, test_size=0.2,
    ↪ random_state=0)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.
    ↪ to_numpy())
X_test = scaler.fit_transform(X_test.to_numpy
    ↪ ())

linreg = LinearRegression()
linreg.fit(X_train, y_train)
y_pred=linreg.predict(X_test)
print(metrics.mean_squared_error(y_test,
    ↪ y_pred))

linreg.coef_

# %% [markdown]
# #### Linear SVR

# %%
from sklearn.svm import LinearSVR

X = reader.get_features("dropout", "median")
y = reader.get_label('total', "sqrt_log")
y=np.power(y,1/2)
X_train, X_test, y_train, y_test =
    ↪ train_test_split(X, y, test_size=0.2,
    ↪ random_state=0)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.
    ↪ to_numpy())
X_test = scaler.fit_transform(X_test.to_numpy
    ↪ ())


svr = LinearSVR(C=0.0005)
svr = svr.fit(X_train, y_train.values.ravel())
y_pred=svr.predict(X_test)

print(metrics.mean_squared_error(y_test,
    ↪ y_pred))
SV=svr.coef_
print("Weights : \n", SV)


# %%
```