# ENGG 680: Introduction to Digital Engineering
# Assignment 2

Adam Smith

October 23, 2022

## Preliminary Data Analysis

We begin by loading the data into a Google Colab noteboook as a pandas DataFrame and checking for missing values. The data appears to contain no `null` values, so there is likely no need to impute any data. Next, we need to encode the categorical attributes. The first attribute we look at is "Community Name". This feature has 300 distinct values, so I originally considered applying a frequency or binary encoding to this feature so as to not introduce too many new features. However, I found that a one-hot encoding whereby we keep the most common $n$ communities and lump the rest into an "Other" category worked the best. After some experimentation, I settled on $n = 80$. The next feature, "Group Category", has only two values, so we simply map Disorder to 0 and Crime to 1. The "Category" feature contains 10 categories with many values, and one category called "1320.131" with a single value. On the Criminal Code of Canada's website, one can find that 1320.131 corresponds to "Dangerous operation of a motor vehicle". So that we don't have a category with only one value, I considered this similar to violence and lumped it in with the "Violence Other (Non-domestic)" category. I then performed a one-hot encoding on this feature with the pandas `get_dummies` function to spread it into 10 new features.

The next feature, "Resident Count", is numerical, so I examined its distribution with a violin plot (fig. 1). The distribution is skewed right, and has an unbiased skew value of 1.05. Unskewed features generally perform better in machine learning tasks, so I applied a square root transformation to this feature in an attempt to reduce the skewness. The resulting distribution (fig. 2) appeared less skewed, and had a better skew value of -0.22.

We will not apply any transformation to the "Year" feature (except for standardization, which will come later).

For the remaining two features, "Sector" and "Month", my original thought was to apply a sin-cos encoding to preserve their cyclic nature. In this encoding, one maps the categories to the unit circle and then creates two new features with the sin and cos functions. This should perform better for deep learning applications, however, I found that it did not perform well for linear regression. This is likely because the trigonometric functions introduce non-linearities into the features which the linear regression cannot model effectively. Thus, I decided to go with a target encoding for these features. In a target encoding, we map each of the feature's values to the mean its corresponding target variable values. Of course, we must only fit the transformer on the training data so that we do not include any information about the test data into the training set. Hence, we apply the train-test split here, setting aside 30% of the data for testing and leaving 70% of it for training. The target encoding can then be done using the `TargetEncoder` class from the `category_encoders` package.

Finally, we examine our target variable, "Crime Count". As shown in the violin plot in fig. 3, our target variable is highly skewed, and has an unbiased skew of 14.85. To remedy this, we can apply a non-linear

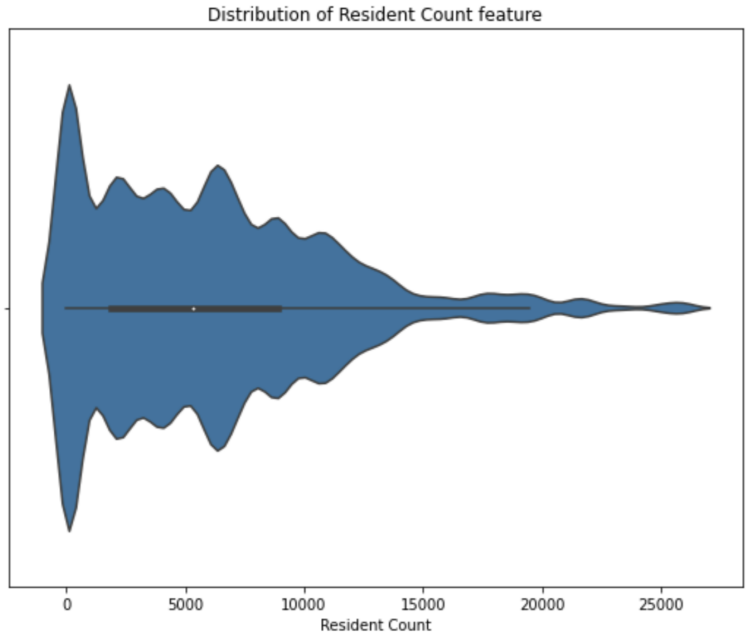


Figure 1: Violin plot showing the distribution of the Resident Count feature.

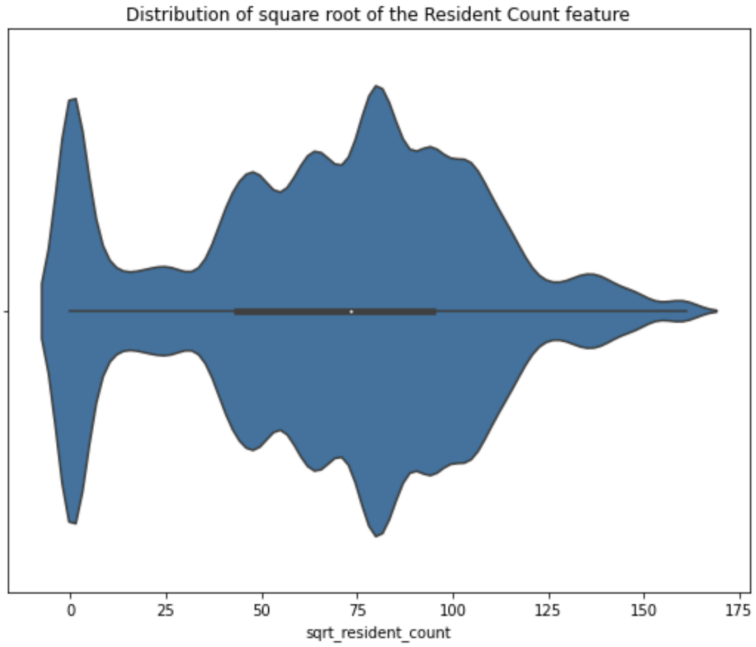


Figure 2: Violin plot showing the distribution of the square root of the Resident Count feature.

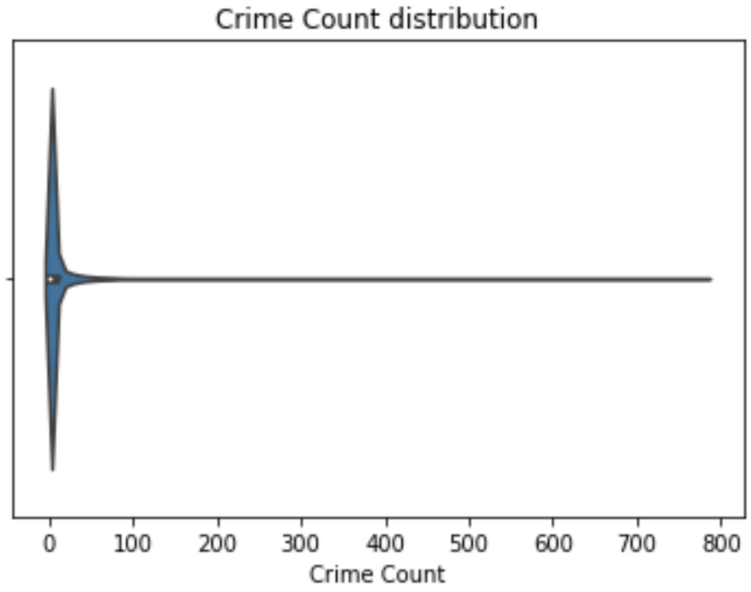


Figure 3: Violin plot showing the distribution of the target variable Crime Count.

transformation to the data. Our algorithm will then predict the transformed value, and we must reverse the transformation afterwards. Through testing, I found that applying a logarithmic transform followed by a square root produced the best results. The distribution of the transformed target variable is shown in fig. 4. Applying

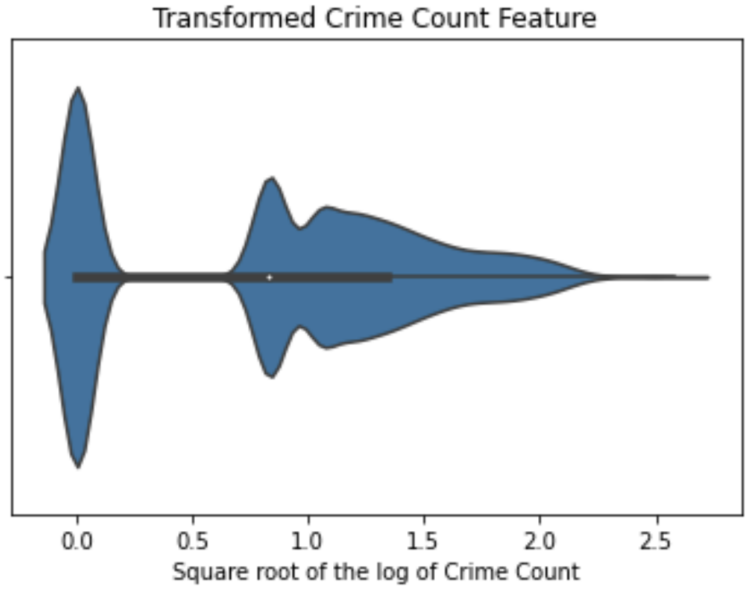


Figure 4: Violin plot showing the distribution of the transformed Crime Count.

this transformation reduced the skew to -0.0024.

## Model Description

Now, we are ready to standardize the data and train our model. We use scikit-learn's `StandardScaler` class to scale the training and testing input data based on the mean and standard deviation of the training data.

We will apply the linear regression model using scikit-learn's `LinearRegression` class. We fit the model on the training data and then use it to predict values for the square root of the logarithm of the crime count. Before analyzing results, we of course have to square and exponentialize the predictions to get the predicted values of the crime count.

## Results

Figures 5 and 6 show plots of the predicted vs. actual results for the testing and training data respectively. To

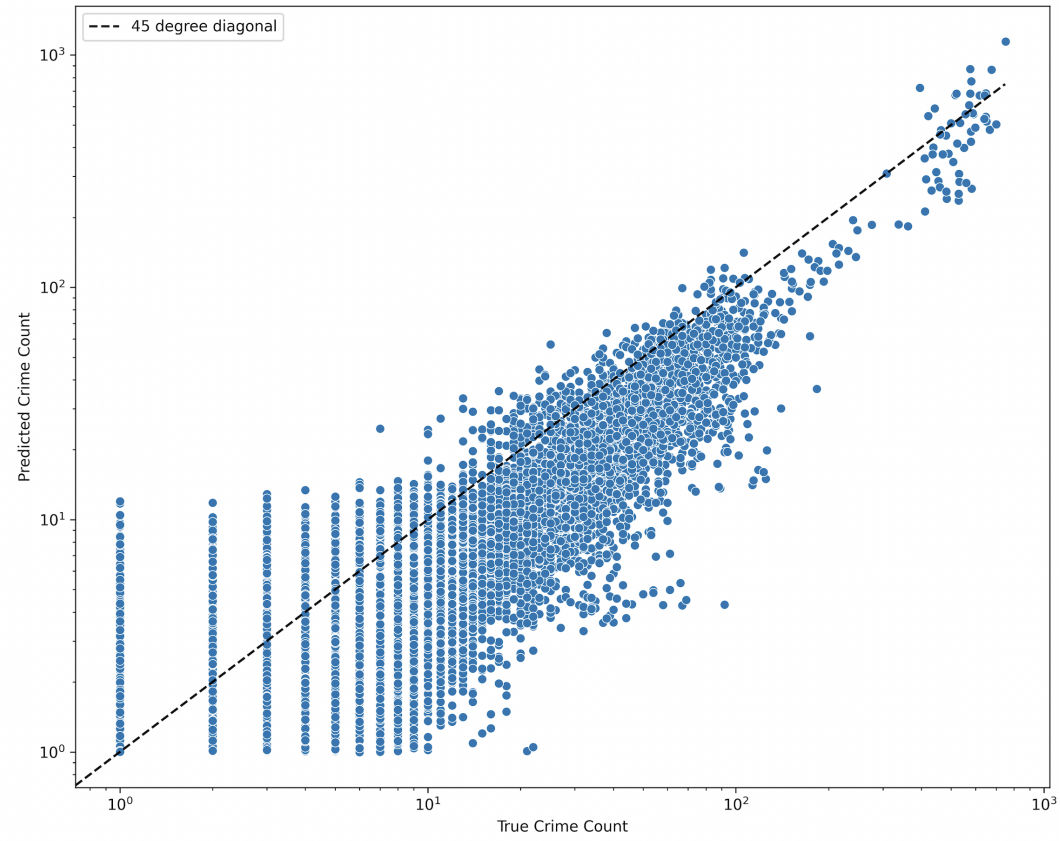


Figure 5: Scatterplot of the predicted vs. actual crime count for the testing dataset. Note the logarithmic axes scaling.

evaluate the model, we obtain a Coefficient of Determination for the test data of $R^2 = 0.839$, and $R^2 = 0.827$ for the training data. The mean squared error for each dataset is $MSE_{test} = 119.79$ and $MSE_{train} = 118.13$. Fig. 7 shows these values as produced by scikit-learn functions. These values are similar, so we can conclude that this model does not display high variance. However, due to the simplicity of the model, we are left with somewhat high bias.

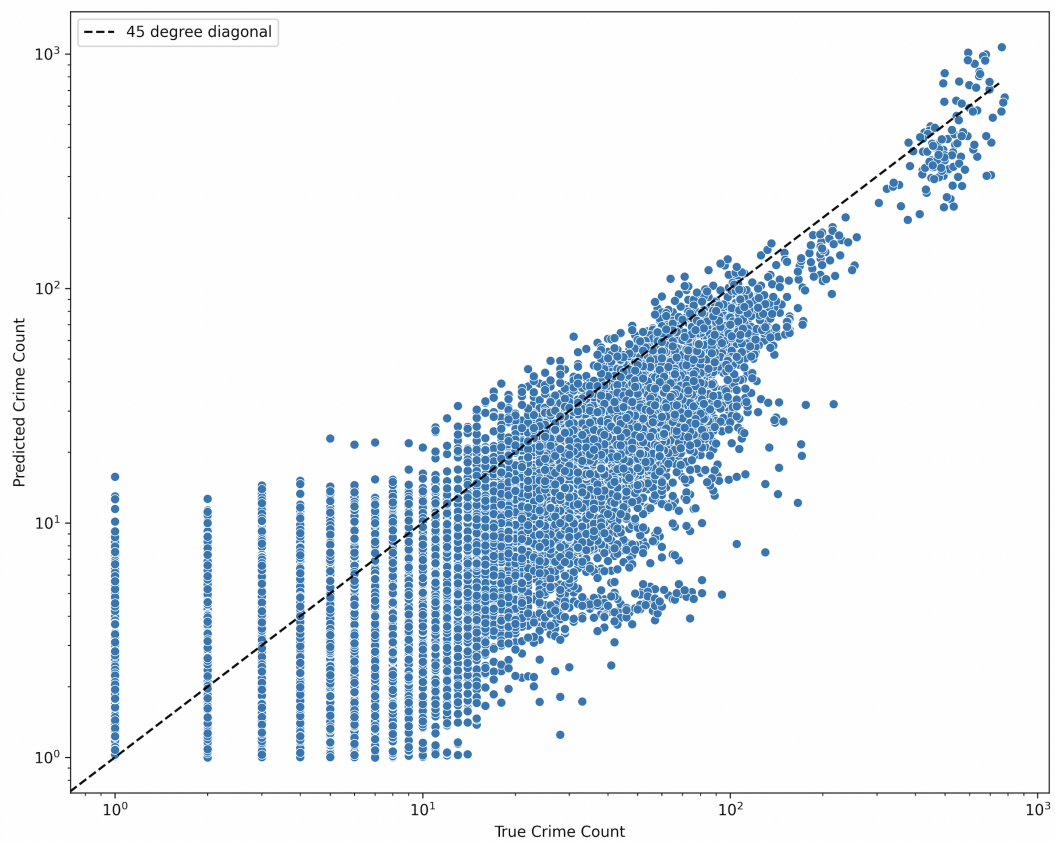


Figure 6: Scatterplot of the predicted vs. actual crime count for the training dataset. Note the logarithmic axes scaling.

```
print(f'MSE for test data: {mean_squared_error(y_test, y_pred)}')
print(f'MSE for train data: {mean_squared_error(y_train, y_pred_train)}')
```

```
MSE for test data: 119.792451451587
MSE for train data: 118.12697687629034
```

```
print(f'MAE for test data: {mean_absolute_error(y_test, y_pred)}')
print(f'MAE for train data: {mean_absolute_error(y_train, y_pred_train)}')
```

```
MAE for test data: 3.6315064215436252
MAE for train data: 3.6251489923535885
```

```
print(f'Coefficient of Determination for test data: {r2_score(y_test, y_pred)}')
print(f'Coefficient of Determination for train data: {r2_score(y_train, y_pred_train)}')
```

```
Coefficient of Determination for test data: 0.838818119136924
Coefficient of Determination for train data: 0.8268441247126399
```

Figure 7: Evaluation metrics for the model as computed by scikit-learn.