In [ ]:
```
import os
```

In [ ]:
```
def is_float(element) -> bool:
    try:
        float(element)
        return True
    except ValueError:
        return False

def is_int(element) -> bool:
    try:
        int(element)
        return True
    except ValueError:
        return False
```

In [ ]:
```
data_path = os.path.join(os.curdir, "CommViolPredUnnormalizedData.txt")

X = []
y = []

with open(data_path, "r") as f:
    for line in f:
        data_values = line.split(",")
        data_item = []
        for data_value in data_values[:-1]:
            if '?' in data_value:
                data_item.append(None)
            elif is_float(data_value):
                data_item.append(float(data_value))
            elif is_int(data_value):
                data_item.append(int(data_value))
            else:
                data_item.append(data_value)
        X.append(data_item)

        target = data_values[-1]
        if '?' in target:
            y.append(None)
        elif is_float(target):
            y.append(float(target))
        else:
            raise Exception("Unexpected target value: {}".format(target))

print(X[0])
print(y[0])
```

```
['BerkeleyHeightstownship', 'NJ', 39.0, 5320.0, 1.0, 11980.0, 3.1, 1.37, 91.7
8, 6.5, 1.88, 12.47, 21.44, 10.93, 11.33, 11980.0, 100.0, 75122.0, 89.24, 1.5
5, 70.2, 23.62, 1.03, 18.39, 79584.0, 29711.0, 30233.0, 13600.0, 5725.0, 2710
1.0, 5115.0, 22838.0, 227.0, 1.96, 5.81, 9.9, 48.18, 2.7, 64.55, 14.65, 28.82,
5.49, 50.73, 3.67, 26.38, 5.22, 4.47, 3.22, 91.43, 90.17, 95.78, 95.81, 44.56,
58.88, 31.0, 0.36, 1277.0, 8.69, 13.0, 20.99, 30.93, 0.93, 1.39, 2.24, 3.3, 8
5.68, 1.37, 4.81, 4.17, 2.99, 3.0, 2.84, 91.46, 0.39, 11.06, 3.0, 64.0, 98.37,
91.01, 3.12, 37.5, 1959.0, 0.0, 0.28, 215900.0, 262600.0, 326900.0, 111000.0,
685.0, 1001.0, 1001.0, 316.0, 1001.0, 23.8, 21.1, 14.0, 11.0, 0.0, 10.66, 53.7
2, 65.29, 78.09, 89.14, None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None, None, None, 6.5, 1845.9, 9.63, None, None,
None, None, 0.0, None, 0.0, 0.0, 0.0, 0.0, 1.0, 8.2, 4.0, 32.81, 14.0, 114.85,
138.0, 1132.08, 16.0, 131.26, 2.0, 16.41, 41.02]
1394.59
```

In [ ]:

```
def get_prop_missing(data):
    column_proportion_missing = []

    for column_index in range(len(data[0])):
        column = [row[column_index] for row in data]
        column_proportion_missing.append(sum([1 if element is None else 0 for

    return column_proportion_missing
```

In [ ]:

```
for i, row in enumerate(X):
    new_entry = row[4:-17]
    X[i] = new_entry

column_proportion_missing = get_prop_missing(X)
print("Proportion of missing values per column:")
print(column_proportion_missing)


column_indexes_to_remove = [column_index for column_index, proportion_missing
print("Columns to remove: {}".format(column_indexes_to_remove))
print("Number of columns to remove: {}".format(len(column_indexes_to_remove)))
```

```
Proportion of missing values per column:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0000514672686230248, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.8451467268623025, 0.84514672
68623025, 0.8451467268623025, 0.8451467268623025, 0.8451467268623025, 0.845146
7268623025, 0.8451467268623025, 0.8451467268623025, 0.8451467268623025, 0.8451
467268623025, 0.8451467268623025, 0.8451467268623025, 0.8451467268623025, 0.84
51467268623025, 0.8451467268623025, 0.8451467268623025, 0.8451467268623025, 0.
0, 0.0, 0.0, 0.8451467268623025, 0.8451467268623025, 0.8451467268623025, 0.845
1467268623025, 0.0, 0.8451467268623025]
Columns to remove: [99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,
111, 112, 113, 114, 115, 119, 120, 121, 122, 124]
Number of columns to remove: 22
```

Here we removed all features that had a large proportion of missing entries. This also means we could have lost information about why a certain feature is missing. To remedy this we could have given all missing values a default value such as -1 and added an additional feature that acts a flag for when that specific feature is missing. We could also have tried to compute

reasonable values to replace the missing ones; however, this approach would have likely introduced bias into the data.

In [ ]:
```
#clean the data
print(len(X), len(X[0]))
for i, row in enumerate(X):
    new_entry = []
    for j, element in enumerate(row):
        if j not in set(column_indexes_to_remove):
            new_entry.append(element)

    X[i] = new_entry

print(len(X), len(X[0]))
```

```
2215 125
2215 103
```

In [ ]:
```
#remove rows with missing targets
new_targets = []
new_data = []
for i, target in enumerate(y):
    if target is not None:
        new_targets.append(target)
        new_data.append(X[i])

X = new_data
y = new_targets

print(len(X), len(y))
```

```
2118 2118
```

In [ ]:
```
column_proportion_missing = get_prop_missing(X)
print("Proportion of missing values per column:")
print(column_proportion_missing)
```

```
Proportion of missing values per column:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.00047214353163361664,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

In [ ]:
```
#remove rows with missing values still
new_data = []
new_targets = []
for i, row in enumerate(X):
    num_nones_in_row = sum([1 if element is None else 0 for element in row])
    if num_nones_in_row == 0:
        new_data.append(row)
        new_targets.append(y[i])

X = new_data
y = new_targets
print(len(X), len(y))
```

```
2117 2117
```

Here we store the cleaned data in a new file for easier use in future models.

In [ ]:
```
#store cleaned code to be used later
import csv

cleaned_data_path = os.path.join(os.curdir, "cleaned_data.csv")

print(len(X), len(X[0]))
for i, row in enumerate(X):
    row.append(y[i])
print(len(X), len(X[0]))

with open(cleaned_data_path, "w+") as f:
    writer = csv.writer(f)
    writer.writerows(X)
```

```
2117 103
2117 104
```

In [ ]:
```
# Run and plot TSNE
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
import sklearn
from IPython.utils import io

def make_and_save_scatter_plot(plot_locations, labels, legend_categories, titl
    sns.set_style("darkgrid")
    fig = plt.figure(figsize=(15, 10))
    ax = fig.add_subplot(111)

    color_list = sns.color_palette("viridis", as_cmap=True)
    sns.scatterplot(
        x=plot_locations[:, 0],
        y=plot_locations[:, 1],
        hue=labels,
        ax=ax,
        size=[5 for _ in range(len(labels))],
        legend= False,
        cmap=color_list,
        hue_norm=(116,12119)
    )
    ax.set_title(title)

    # legend_handels = []
    # for i, category in enumerate(legend_categories):
    #     legend_handels.append(matplotlib.patches.Patch(color= color_list[i],
    # ax.legend(handles=legend_handels)

    # plt.savefig(save_path)
    plt.show()
    # plt.close()

from sklearn.decomposition import PCA

def create_pca_plot(embeddings, labels, legend_categories, title_subject, save
    pca = PCA(n_components=2)
    pca.fit(embeddings, labels)
```

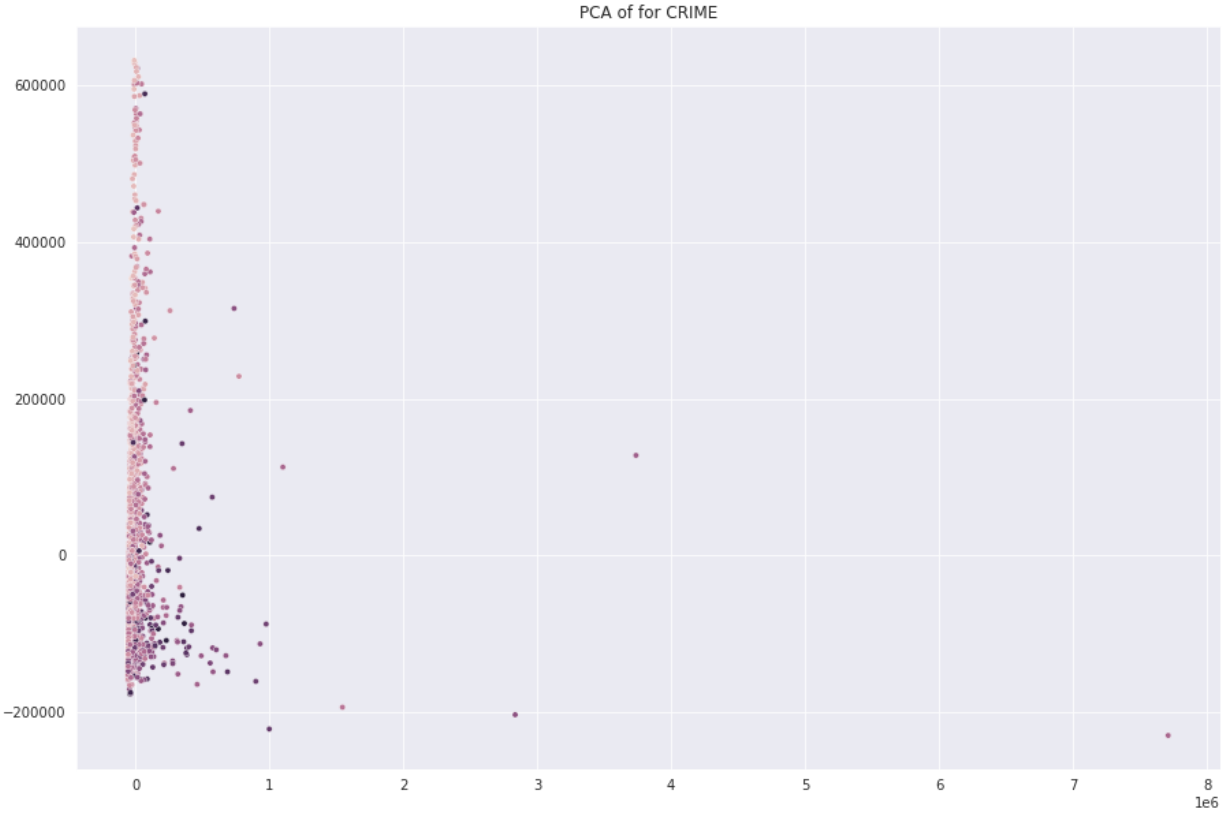
```
    plot_locations = pca.transform(embeddings)

    make_and_save_scatter_plot(plot_locations, labels, legend_categories, f"PC

from sklearn.manifold import TSNE

def create_tsne_plot(embeddings, labels, legend_categories, title_subject, sav
    with io.capture_output() as captured:
        tsne = TSNE(n_components=2, learning_rate='auto', perplexity=(len(embe
        plot_locations = tsne.fit_transform(embeddings)

        make_and_save_scatter_plot(plot_locations, labels, legend_categories,
```

In [ ]:
```
X = []
y= []

data_path = os.path.join(os.path.curdir, "cleaned_data.csv")
with open(data_path) as f:
    for line in f:
        data = line.split(',')
        data_item = [float(x) for x in data[2:-1]]
        X.append(data_item)
        y.append(float(data[-1]))

print(max(y))
print(min(y))
print(sum(y)/len(y))
```

```
27119.76
116.79
4907.080949456779
```

We wanted to get a better sense of the data and create a visual plot. This is basically impossible with our dataset wich is 100 dimensions. To solve this issues we use both PCA and TSNE to create 2D plots of the dataset.

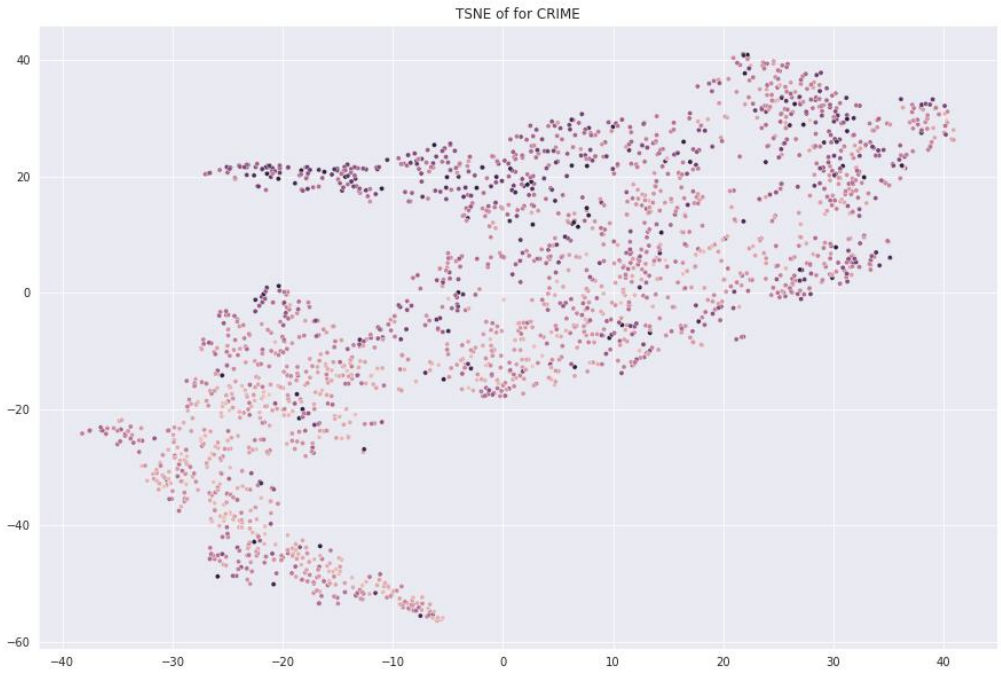
In [ ]:
```
import torch
create_pca_plot(X, y, [], "CRIME", "./graphs")
```



In [ ]:
```
create_tsne_plot(torch.tensor(X), y, [], "CRIME","./graphs")
```

In [ ]:
```
from IPython import display
display.Image("./graphs/tsne.jpg")
```

Out[ ]:



In [ ]:
```
from sklearn.neural_network import MLPRegressor
import os
```

In [ ]:
```
# Load the data
X = []
y= []

data_path = os.path.join(os.path.curdir, "cleaned_data.csv")
with open(data_path) as f:
    for line in f:
        data = line.split(',')
        data_item = [float(x) for x in data[:-1]]
        X.append(data_item)
        y.append(float(data[-1]))

print(len(X), len(X[0]))
print(len(y), y[0])

print(X[0])
```

```
2117 103
2117 1394.59
[1.0, 11980.0, 3.1, 1.37, 91.78, 6.5, 1.88, 12.47, 21.44, 10.93, 11.33, 11980.
0, 100.0, 75122.0, 89.24, 1.55, 70.2, 23.62, 1.03, 18.39, 79584.0, 29711.0, 30
233.0, 13600.0, 5725.0, 27101.0, 5115.0, 22838.0, 227.0, 1.96, 5.81, 9.9, 48.1
8, 2.7, 64.55, 14.65, 28.82, 5.49, 50.73, 3.67, 26.38, 5.22, 4.47, 3.22, 91.4
3, 90.17, 95.78, 95.81, 44.56, 58.88, 31.0, 0.36, 1277.0, 8.69, 13.0, 20.99, 3
0.93, 0.93, 1.39, 2.24, 3.3, 85.68, 1.37, 4.81, 4.17, 2.99, 3.0, 2.84, 91.46,
0.39, 11.06, 3.0, 64.0, 98.37, 91.01, 3.12, 37.5, 1959.0, 0.0, 0.28, 215900.0,
262600.0, 326900.0, 111000.0, 685.0, 1001.0, 1001.0, 316.0, 1001.0, 23.8, 21.
1, 14.0, 11.0, 0.0, 10.66, 53.72, 65.29, 78.09, 89.14, 6.5, 1845.9, 9.63, 0.0]
```

We later use Mean Squared Error to evalutate our models. Mean Squared Error is heavily influenced by outliers, and our target appears to be skewed. The mean value is ~4000 but the max value is ~27000. To help remove skew and normalize the data we remove all data entries with a target more than 3 standard deviations away from the mean.

We also previously tried taking the log of the target. This corrected the skew and problems with the MSE, but caused our models to perform more poorly. Thus, for our final models we did not transform the target.

In [ ]:
```
# Transform the target
import numpy as np

std_y = np.std(y)
mean_y = sum(y)/len(y)

print(std_y, mean_y)

non_outlier_x = []
non_outlier_y = []
for data_values, target in zip(X, y):
    if target < (mean_y + (std_y*3)):
        non_outlier_x.append(data_values)
        non_outlier_y.append(target)
```

```
X= non_outlier_x
y = non_outlier_y
# y = [np.log(x) for x in y]

std_y = np.std(y)
mean_y = sum(y)/len(y)

print(std_y, mean_y)
```

```
2739.1879600147977 4907.080949456779
2446.7344360199154 4780.3492164357385
```

Next to geta good estimate of our models performance in the real world we created and ran K-fold Cross validation on all of our models. We used 6 folds, meaning the results we display hear after are the average r^2 and MSE of the 6 models of a given type.

K-fold Cross Validation lets us train and test on all the data while still being able to not fit on the test data.

In [ ]:
```
# Setup folds for k-fold cross validation
import random
k = 6
fold_size = len(X) // k
fold_diff = len(X) - (fold_size * k)

folds_x = []
folds_y = []
data_to_shuffle = list(zip(X, y))
random.shuffle(data_to_shuffle)
X, y = zip(*data_to_shuffle)
for fold_id in range(k):
    start_index = fold_id * fold_size
    end_index = (fold_id + 1) * fold_size

    if fold_id == k - 1:
        end_index += fold_diff

    folds_x.append(X[start_index:end_index])
    folds_y.append(y[start_index:end_index])

print(len(folds_x), len(folds_x[0]), len(folds_x[0][0]))
print(len(folds_y), len(folds_y[0]))
```

```
6 348 103
6 348
```

In [ ]:
```
import numpy as np

def split_into_sets(folds_x, folds_y, fold_id):
    x_train = folds_x[:fold_id] + folds_x[fold_id + 1:] if fold_id < len(folds
    x_train = [item for sublist in x_train for item in sublist]

    y_train = folds_y[:fold_id] + folds_y[fold_id + 1:] if fold_id < len(folds
    y_train = [item for sublist in y_train for item in sublist]

    x_test = folds_x[fold_id]
    y_test = folds_y[fold_id]
```

```
    return x_train, y_train, x_test, y_test
```

In [ ]:
```
# Run simple linear regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from tqdm import tqdm

coef_of_determination = []
mse = []
for fold_id in tqdm(range(k)):
    x_train, y_train, x_test, y_test = split_into_sets(folds_x, folds_y, fold_
    model = LinearRegression().fit(x_train, y_train)

    coef_of_determination.append(model.score(x_test, y_test))
    mse.append(mean_squared_error(y_test, model.predict(x_test)))

print(f"The average coefficient of determination is {np.mean(coef_of_determina
print(f"The average MSE is {np.mean(mse)}")
print("The intercept and coeficients of the last model are:")
print(model.intercept_, model.coef_)
```

```
100%|██████████| 6/6 [00:00<00:00, 62.09it/s]
The average coefficient of determination is 0.5955236225598252
The average MSE is 2404989.272144063
The intercept and coeficients of the last model are:
-7605.955670974752 [-1.40523569e+01  2.46158746e-02 -1.32166400e+03  1.2833914
3e+00
  4.43511821e+00  2.38306485e+01 -7.01467422e+00  1.85395549e+02
  7.03758297e+01 -2.44851843e+02  2.40210721e+01 -2.18012398e-02
  6.05741651e+00 -6.10297219e-02  4.47406623e+01 -2.84927581e+01
  9.07839710e+00  1.12925422e+02 -8.77360818e+00 -6.12643308e+01
  5.03093609e-02 -1.83812384e-01  2.19277071e-01 -5.80792197e-03
  8.52492406e-04 -1.50651323e-03  1.25699079e-02 -2.62746297e-03
  1.66828435e-02  4.74563208e+01 -1.00852770e+01 -3.28738871e+01
 -5.01930820e+01  1.02271614e+01  5.27278879e+01 -2.15484960e+01
 -9.78145655e+00  3.76552795e+01  6.95971019e+01  5.59363498e+02
  7.23734850e+01  5.06691344e+02 -8.85796187e+02 -3.20296345e+03
  7.76310082e+00 -7.86381498e+01  5.98010633e+00 -8.69904209e+00
  7.35508801e+00 -2.84118986e+01 -5.86195215e-02  2.02079670e+01
 -7.80084910e-03  1.79749119e+01 -4.26377633e+01  1.43503384e+01
  2.42451228e+01 -2.47421067e+02  2.32135588e+02 -1.39415146e+02
 -1.21345191e+02 -7.04225352e+00  4.38851248e+01  1.44775643e+02
 -6.20083163e+01  2.22023352e+03  1.90588030e+03 -1.57734967e+03
 -2.02150110e+02 -1.93022593e+00 -5.71905154e+00 -5.61786224e+01
 -2.26665747e-02 -3.14968411e+01  1.90519330e+02 -1.68211141e+00
 -1.11668525e+01  7.77322338e+00  2.26865567e+00  1.20266045e+02
 -6.52265101e-03  7.05131882e-03 -6.02844970e-03  4.93923712e-04
 -3.09476593e-01 -8.83783974e-01 -3.72737023e-01 -6.32604330e-02
  1.15223511e+00  2.54909444e+01 -6.82316984e+01 -2.62366912e+01
 -5.15503453e-02  9.11395161e-01  9.41069275e+01 -6.33973423e+00
 -1.82037241e+01  4.68799523e+00  8.26456881e+00 -6.64343379e-01
 -9.01066175e-02 -5.15080259e+00  5.67930264e+01]
```

Working with Dr.Page we noticed that our dataset has a high level of colinearity. To fix this and reduce the number of features while mainting all of the relevant information we transformed the dataset to a new basis using PCA. We tried a few different numbers of PCA components but 35 seemed to be the perfect balance between performance and number of features.

In [ ]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=35)
pca.fit(X, y)
X = pca.transform(X)

print(len(X), len(X[0]))
```

```
2093 35
```

In [ ]:
```
# Setup folds for k-fold cross validation
import random
k = 6
fold_size = len(X) // k
fold_diff = len(X) - (fold_size * k)

folds_x = []
folds_y = []
data_to_shuffle = list(zip(X, y))
random.shuffle(data_to_shuffle)
X, y = zip(*data_to_shuffle)
for fold_id in range(k):
    start_index = fold_id * fold_size
    end_index = (fold_id + 1) * fold_size

    if fold_id == k - 1:
        end_index += fold_diff

    folds_x.append(X[start_index:end_index])
    folds_y.append(y[start_index:end_index])

print(len(folds_x), len(folds_x[0]), len(folds_x[0][0]))
print(len(folds_y), len(folds_y[0]))
```

```
6 348 35
6 348
```

In [ ]:
```
# Run simple linear regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from tqdm import tqdm

coef_of_determination = []
mse = []
for fold_id in tqdm(range(k)):
    x_train, y_train, x_test, y_test = split_into_sets(folds_x, folds_y, fold_
    model = LinearRegression().fit(x_train, y_train)

    coef_of_determination.append(model.score(x_test, y_test))
    mse.append(mean_squared_error(y_test, model.predict(x_test)))

print(f"The average coefficient of determination is {np.mean(coef_of_determina
print(f"The average MSE is {np.mean(mse)}")
print(model.intercept_, model.coef_)
```

```
100%|██████████| 6/6 [00:00<00:00, 399.70it/s]
```

```
The average coefficient of determination is 0.5463152552176137
The average MSE is 2715190.6926644356
4802.216571955404 [ 1.05811920e-03 -5.48429269e-03 -1.02921346e-02 -1.98846837
e-02
 -3.06319947e-02  5.90302064e-02  1.33027313e-02  1.75392992e-02
  3.79780965e-02 -3.08912688e-02  3.12427912e-02  4.91303003e-02
  4.97782473e-02  2.10418001e-01  2.52654805e-02  2.95999644e-02
  1.15178759e-01 -2.76923321e-01  7.73830840e-01 -4.40896236e-01
  1.68767720e+00 -2.53184149e-01 -6.37010582e-01 -4.39681960e-01
  7.86518902e+00  6.15011358e+00  6.32293846e+00  1.07747621e+01
  9.41437841e+00  1.25846639e+01  2.98045543e+01 -1.12411195e+01
 -2.20114859e+01  5.48681750e+00 -3.70913138e+00]
```

In [ ]:

```
from sklearn.linear_model import LassoLars
from IPython.utils import io

coef_of_determination = []
for fold_id in tqdm(range(k)):
    x_train, y_train, x_test, y_test = split_into_sets(folds_x, folds_y, fold_
    with io.capture_output() as captured:
        model = LassoLars().fit(x_train, y_train)

    coef_of_determination.append(model.score(x_test, y_test))

print(f"The average coefficient of determination is {np.mean(coef_of_determina
print(model.intercept_, model.coef_)
```

```
100%|██████████| 6/6 [00:00<00:00, 296.30it/s]
The average coefficient of determination is 0.5486012413180624
4795.515671041799 [ 8.72424103e-04 -5.15192694e-03 -8.63289309e-03 -1.66104011
e-02
 -2.68601863e-02  5.68738987e-02  8.17052350e-03  1.19191357e-02
  3.26319191e-02 -2.53733330e-02  2.43241901e-02  4.02942679e-02
  4.03879022e-02  1.94335663e-01  0.00000000e+00  0.00000000e+00
  8.08387547e-02 -2.59112149e-01  7.05351559e-01 -2.05901007e-01
  1.36867460e+00  0.00000000e+00 -5.88283706e-02  0.00000000e+00
  6.54097260e+00  4.61758937e+00  4.17940168e+00  8.81751185e+00
  6.48142908e+00  9.96329814e+00  2.68883441e+01 -7.16457042e+00
 -1.80506253e+01  1.31569854e+00  0.00000000e+00]
```

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchmetrics import R2Score

class CrimeData(torch.utils.data.Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X)
        self.y = torch.tensor(y)
        self.len = self.X.shape[0]

    def __getitem__(self, index):
        return {"data": self.X[index], "target": self.y[index]}

    def __len__(self):
        return self.len

class Net(torch.nn.Module):
```

```
    def __init__(self, num_features):
        super(Net, self).__init__()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(num_features, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1028),
            nn.ReLU(),
            nn.Linear(1028, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 1),
            nn.ReLU()
        )

    def forward(self, x):
        logits = self.linear_relu_stack(x)
        return logits
```

We seem to have lost the results for our deep learning model, but before losing them the model was perfomring very simalarly to the regressions we ran earlier. However, on two of the 6 folds we would get a negative R^2 score which brought our average down quite a bit. We still don't know why that was happening, but given more time is something we would definetly look into.

In [ ]:
```
# Setup folds for k-fold cross validation
from numpy import average
from IPython.utils import io
from tqdm import tqdm

loss_fn = torch.nn.MSELoss()
r2score = R2Score().to("cuda")
average_test_r2 = []
num_epochs = 80

for fold_id in tqdm(range(k)):
    x_train, y_train, x_test, y_test = split_into_sets(folds_x, folds_y, fold_
    train_loader = torch.utils.data.DataLoader(dataset=CrimeData(x_train, y_tr
    test_loader = torch.utils.data.DataLoader(dataset=CrimeData(x_test, y_test

    model = Net(len(x_train[0])).to("cuda")
    optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)

    with io.capture_output() as captured:
        for epoch in range(num_epochs):
            # Train model
            model.train()
            training_losses = []
            train_r2_scores = []
            p_bar = tqdm(train_loader, desc="Training")
            for batch in p_bar:
                data = batch['data'].to('cuda').float()
                target = batch['target'].to('cuda').float()
                optimizer.zero_grad()
                logits = model(data)
                logits = logits[:,0]
                loss = loss_fn(logits, target)
```

```
                loss.backward()
                optimizer.step()

                training_losses.append(loss.item())
                train_r2_scores.append(r2score(logits, target))
                p_bar.set_postfix({
                    "loss" : f"{sum(training_losses)/len(training_losses):.4f}
                    "R^2" : f"{sum(train_r2_scores)/len(train_r2_scores):.4f}"
                })

    # Test model
    model.eval()
    test_losses = []
    test_r2_scores = []
    with torch.no_grad():
        p_bar = tqdm(test_loader, desc="Test")
        for batch in p_bar:
            data = batch['data'].to('cuda').float()
            target = batch['target'].to('cuda').float()
            logits = model(data)
            logits = logits[:,0]
            loss = loss_fn(logits, target)

            test_losses.append(loss.item())
            test_r2_scores.append(r2score(logits, target))
            p_bar.set_postfix({
                "loss": f"{sum(test_losses)/len(test_losses):.4f}",
                "R^2" : f"{sum(test_r2_scores)/len(test_r2_scores):.4f}"
            })
        average_test_r2.append(sum(test_r2_scores)/len(test_r2_scores))

print(f"Average test R^2: {sum(average_test_r2)/len(average_test_r2)}")
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/home/dallin/byu/fall_2022/data_capstone/crime_experiments/basic_nueral.ipynb
 Cell 15 in <cell line: 11>()
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=33'>34</a>
optimizer.step()
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=35'>36</a>
training_losses.append(loss.item())
---> <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=36'>37</a>
train_r2_scores.append(r2score(logits, target))
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=37'>38</a>
p_bar.set_postfix({
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=38'>39</a>
"loss" : f"{sum(training_losses)/len(training_losses):.4f}",
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=39'>40</a>
"R^2" : f"{sum(train_r2_scores)/len(train_r2_scores):.4f}"
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=40'>41</a>
})
     <a href='vscode-notebook-cell:/home/dallin/byu/fall_2022/data_capstone/cr
ime_experiments/basic_nueral.ipynb#X14sZmlsZQ%3D%3D?line=42'>43</a> # Test mod
el

File ~/miniconda3/envs/crime_experiments/lib/python3.10/site-packages/torch/n
n/modules/module.py:1130, in Module._call_impl(self, *input, **kwargs)
   1126 # If we don't have any hooks, we want to skip the rest of the logic in
   1127 # this function, and just call forward.
   1128 if not (self._backward_hooks or self._forward_hooks or self._forward_p
re_hooks or _global_backward_hooks
   1129         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1130     return forward_call(*input, **kwargs)
   1131 # Do not call functions when jit is used
   1132 full_backward_hooks, non_full_backward_hooks = [], []

File ~/miniconda3/envs/crime_experiments/lib/python3.10/site-packages/torchmet
rics/metric.py:245, in Metric.forward(self, *args, **kwargs)
    243     self._forward_cache = self._forward_full_state_update(*args, **kwa
rgs)
    244 else:
--> 245     self._forward_cache = self._forward_reduce_state_update(*args, **k
wargs)
    247 return self._forward_cache

File ~/miniconda3/envs/crime_experiments/lib/python3.10/site-packages/torchmet
rics/metric.py:310, in Metric._forward_reduce_state_update(self, *args, **kwar
gs)
    308 # calculate batch state and compute batch value
    309 self.update(*args, **kwargs)
--> 310 batch_val = self.compute()
    312 # reduce batch and global state
    313 self._update_count = _update_count + 1

File ~/miniconda3/envs/crime_experiments/lib/python3.10/site-packages/torchmet
rics/metric.py:531, in Metric._wrap_compute.<locals>.wrapped_func(*args, **kwa
```

```
rgs)
    523 # compute relies on the sync context manager to gather the states acro
ss processes and apply reduction
    524 # if synchronization happened, the current rank accumulated states wil
l be restored to keep
    525 # accumulation going if ``should_unsync=True``,
    526 with self.sync_context(
    527     dist_sync_fn=self.dist_sync_fn,  # type: ignore
    528     should_sync=self._to_sync,
    529     should_unsync=self._should_unsync,
    530 ):
--> 531     value = compute(*args, **kwargs)
    532     self._computed = _squeeze_if_scalar(value)
    534 return self._computed

File ~/miniconda3/envs/crime_experiments/lib/python3.10/site-packages/torchmet
rics/regression/r2.py:126, in R2Score.compute(self)
    124 def compute(self) -> Tensor:
    125     """Computes r2 score over the metric states."""
--> 126     return _r2_score_compute(
    127         self.sum_squared_error, self.sum_error, self.residual, self.to
tal, self.adjusted, self.multioutput
    128     )

File ~/miniconda3/envs/crime_experiments/lib/python3.10/site-packages/torchmet
rics/functional/regression/r2.py:79, in _r2_score_compute(sum_squared_obs, sum
_obs, rss, n_obs, adjusted, multioutput)
     57 """Computes R2 score.
     58
     59 Args:
   (...)
     76     tensor([0.9654, 0.9082])
     77 """
     78 if n_obs < 2:
---> 79     raise ValueError("Needs at least two samples to calculate r2 scor
e.")
     81 mean_obs = sum_obs / n_obs
     82 tss = sum_squared_obs - sum_obs * mean_obs

ValueError: Needs at least two samples to calculate r2 score.
```

In our next analysis, we wanted to use what we had been learning about in class regarding decision trees. Thus, we reoorgainzed the data to be friendly to a decision tree. Namely we simply cut the target variable into 5 different "bins" (very low, low, medium, high, very high) so that we would be able to categorize each of the deicisons into it. No attempt at pruning the trees were made since there were so many variables. The primary difference between these models and the others that we created is that this one did not drop any columns due to missing values--we simply just averaged the values surrounding the missing values and replaces the NaNs with the average. This is because, after running the models several times both ways, we discovered that the accuracy was pretty close, and decided that more data was better than less data, so we kept the averages.

```
target = pd.cut(target.nonViolPerPop, bins=5, labels=np.arange(5))
```

Next, we built the models. While these models are admitadly simple, they seemed to be fairly accurate in their decisions. As we had covered in class, there were multiple criteria with which we could build the tree. To see which was the most important, we built both trees with randomly selected train and test data (80-20), then built 50 models for both criteria.

Build the Tree Using Entropy as Criterion

```
model = tree.DecisionTreeClassifier(criterion = 'entropy')
score_ent = []
for i in range(50):
  target_train, target_test, inputs_train, inputs_test=train_test_split(target,inputs,test_si
  model.fit(inputs_train, target_train)
  score_ent.append(model.score(inputs_test, target_test))
print(np.array(score_ent).mean(),np.array(score_ent).std())
```

```
0.8037735849056605 0.01524833007098547
```

Build the Tree Using Gini as Criterion

```
model = tree.DecisionTreeClassifier(criterion = 'gini')
score_gini = []
for i in range(50):
  target_train, target_test, inputs_train, inputs_test=train_test_split(target,inputs,test_si
  model.fit(inputs_train, target_train)
  score_gini.append(model.score(inputs_test, target_test))
print(np.array(score_gini).mean(),np.array(score_gini).std())
```

```
0.7970283018867925 0.015008674956794288
```

After Doing a T-test, it seems that entropy is better for this test with p-value = 0.0281. So, it seems that Entropy is the better criterion for this data. It predicted with a mean accuracy of 0.8038.

Exploring the data, we find that the most "important" factors ended up being first pctKids2Par, second houseVacant, followed by pct2Par and pctMaleDivorc which seemed to be equally represented. This seems to show that family is one of the deciding factors in how to predict the non-violent crime levels in an area. How interesting!!