DS4420 FINAL PROJECT

Ryan Liang

In [1]:
```
import math
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline

# For Preprocessing and Data manipulation
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

# Model Imports
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression

# Metric Imports

from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    precision_score,
    recall_score,
    f1_score,
    log_loss
)
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import mean_squared_error
from sklearn.inspection import permutation_importance
```

DATA IMPORT

In [2]:
```
data = pd.read_csv("data/MassShootingsDatasetVer5.csv")
```

In [3]:
```
# These columns weren't relevant or useful to the goal of the project
data = data.drop(columns=["Employed at", "S#"])
```

In [4]:
```
data
```

Out[4]:

	Title	Location	Date	Incident Area	Open/Close Location	Target	Cause	Summary
0	Texas church mass shooting	Sutherland Springs, TX	11/5/17	Church	Close	random	unknown	Devin Patrick Kelley, 26, an ex-air force offi...
1	Walmart shooting in suburban Denver	Thornton, CO	11/1/17	Wal-Mart	Open	random	unknown	Scott Allen Ostrem, 47, walked into a Walmart ...
2	Edgewood businees park shooting	Edgewood, MD	10/18/17	Remodeling Store	Close	coworkers	unknown	Radee Labeeb Prince, 37, fatally shot three pe...
3	Las Vegas Strip mass shooting	Las Vegas, NV	10/1/17	Las Vegas Strip Concert outside Mandala Bay	Open	random	unknown	Stephen Craig Paddock, opened fire from the 32...
4	San Francisco UPS shooting	San Francisco, CA	6/14/17	UPS facility	Close	coworkers	NaN	Jimmy Lam, 38, fatally shot three coworkers an...
...	...	...	...	...	...	...	...	...
318	Clara Barton Elementary School	Chicago, Illinois	1/17/74	Clara Barton Elementary School	Close	Teachers	anger	On January 17, 1974, a 14-year-old student ent...
319	New Orleans Police Shootings	New Orleans, Louisiana	12/31/72	NaN	NaN	random	psycho	On New Year's Eve in 1972, a 23-year-old ex-Na...
320	St. Aloysius Church	Spokane, Washington	11/11/71	Church	Close	random	terrorism	On November 11, 1971, a former MIT student ent...
321	Rose-Mar College of Beauty	Mesa, Arizona	11/12/66	Rose-Mar College of Beauty	Close	random	terrorism	On November 12, 1966, an 18-year-old high scho...
322	University of Texas at Austin	Austin, Texas	8/1/66	University of Texas	Close	random	terrorism	On August 1, 1966, a 25-year-old engineering s...

323 rows × 19 columns

Data Processing

In [5]:
```
# For this project, I will encode the following as variables:
# - Gender as a binary variable where 0 = male and 1 = female. (a variab
le 2 means unknown)
# - Mental Health Issues as a binary variable where 0 = no health issues
and 1 = health issues. (a variable 2 means unknown)
# - Race as a variable where:
# 0 = white, 1 = black, 2 = asian, 3 = latino, 4 = native american, 5 =
 2+ races, 6 = unknown/some other race
# -------------------------------------------------------

races = []
genders = []
health = []
# ages = []

for column in data:
    if column == "Race":
        for row in data[column]:
            if row not in races:
                races.append(row)
    if column == "Gender":
        for row in data[column]:
            if row not in genders:
                genders.append(row)
    if column == 'Mental Health Issues':
        for row in data[column]:
            if row not in health:
                health.append(row)
#     if column == 'Age':
#         for row in data[column]:
#             if row not in ages:
#                 ages.append(row)

races_df = []
genders_df = []
health_df = []
# ages_df = []

# ----- (1) ENCODING RACE AS A VARIABLE -----

for i in range(len(data)):
    race_index = races.index(data['Race'][i])

    if race_index == 8 or race_index == 13 or race_index == 15:
        races_df.append(0)

    elif race_index == 7 or race_index == 12 or race_index == 16:
        races_df.append(1)

    elif race_index == 9 or race_index == 17:
        races_df.append(2)

    elif race_index == 14:
        races_df.append(4)

    elif race_index == 11:
```

```
        races_df.append(5)

    elif race_index == 3 or race_index == 5 or race_index == 6 or race_i
ndex == 10:
        races_df.append(6)

    else:
        races_df.append(race_index)

# ----- (2) ENCODING GENDER AS A VARIABLE -----

for i in range(len(data)):
    gender_index = genders.index(data['Gender'][i])

    if gender_index == 2:
        genders_df.append(0)

    elif gender_index == 1 or gender_index == 3 or gender_index == 4:
        genders_df.append(2)

    elif gender_index == 5:
        genders_df.append(1)

    else:
        genders_df.append(gender_index)

# ----- (3) ENCODING MENTAL HEALTH AS A VARIABLE -----

for i in range(len(data)):
    health_index = health.index(data['Mental Health Issues'][i])

    if health_index == 2:
        health_df.append(1)

    elif health_index == 1 or health_index == 3 or health_index == 4:
        health_df.append(2)

    else:
        health_df.append(health_index)

# ----- (4) CLEANING AGE AS A VARIABLE -----

# for i in range(len(data)):
#     age_index = ages.index(data['Age'][i])

#     if age_index == 10:
#         ages_df.append(0)

#     else:
#         ages_df.append(age_index)


# project_dict = {"Race":races_df, "Gender":genders_df, "Mental Health I
ssues":health_df, "Age":ages_df, "Total victims":data["Total victims"]}
project_dict = {"Race":races_df, "Gender":genders_df, "Mental Health Iss
ues":health_df, "Total victims":data["Total victims"]}
project_df = pd.DataFrame(data=project_dict)
```

In [6]:
```
project_df
```

Out[6]:

	Race	Gender	Mental Health Issues	Total victims
0	0	0	0	46
1	0	0	0	3
2	1	0	0	6
3	0	0	2	585
4	2	0	1	5
...	...	...	...	...
318	6	0	1	4
319	1	0	1	22
320	0	0	1	5
321	0	0	1	6
322	0	0	1	48

323 rows × 4 columns

In [7]:
```
# -- preprocessing / scaling -- #

# introduce the StandardScaler model to normalize/scale the data
scaler = StandardScaler()
```

PROBLEM 1

In [8]:
```
# FEATURE PROBLEM 1 - predicting whether the shooter was male or female
# This will be considered a Classification task. Classify/predict whethe
r the shooter was one or the other (binary)
```

In [9]:
```
# -- preprocessing / scaling (cont.) -- #

# create a copy of the project df for this specific problem
problem1_df = project_df

# get the indicies where the Gender was unknown
indicies = problem1_df[problem1_df['Gender'] == 2].index

# dropping values where Gender == 2 (the gender was unknown)
problem1_df = problem1_df.drop(indicies)

# reset the indicies so we don't get NaN values
problem1_df = problem1_df.reset_index(drop=True)

# transform the training and testing data to scale. drop non-numerical c
olumns
scaled_data = scaler.fit_transform(problem1_df.drop(['Gender'], axis = 1
))

# turn transformed/scaled data back into dataframes with header columns
scaled_df = pd.DataFrame(scaled_data, columns=problem1_df.columns.drop([
'Gender']))
scaled_df['Gender'] = problem1_df['Gender']

# assign the features and target to the approriate columns/variables
features = scaled_df.drop(['Gender'], axis = 1)
target = scaled_df['Gender']
```

In [10]:
```
# For context, the data contains about 98% male shooters (292/297) and
 2% female shooters
problem1_df['Gender'].value_counts()
```

Out[10]:
```
0    292
1      5
Name: Gender, dtype: int64
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(features, target, te
st_size=0.65)

# scale the features
X_train = StandardScaler().fit_transform(X_train.values)
X_test = StandardScaler().fit_transform(X_test.values)
```

Model 1: Naive Bayes

In [12]:
```
# ----- MODEL 1: Naive Bayes ----- #

print("\n----- Naive Bayes -----")

nb = GaussianNB()
nb.fit(X_train, y_train)

# TRAINING
print("On Training Data:")
train_predict_label = nb.predict(X_train)

c_matrix = confusion_matrix(y_train, train_predict_label)

tp = c_matrix[1][1]
tn = c_matrix[0][0]

accuracy = (tp + tn) / (len(train_predict_label))
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}\n".format(1-accuracy))

# TESTING
print("On Testing Data:")
test_predict_label = nb.predict(X_test)

c_matrix = confusion_matrix(y_test, test_predict_label)

tp = c_matrix[1][1]
tn = c_matrix[0][0]

accuracy = (tp + tn) / len(test_predict_label)
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}\n".format(1-accuracy))
```

```
----- Naive Bayes -----
On Training Data:

The accuracy is: 0.9902912621359223
The error is: 0.009708737864077666

On Testing Data:

The accuracy is: 0.979381443298969
The error is: 0.020618556701030966
```

Model 2: Logistic Regression

In [13]:
```
# ----- MODEL 2: Logistic Regression ----- #

lr = LogisticRegression()
lr.fit(X_train, y_train)
coeff_df = pd.DataFrame(lr.coef_.T, problem1_df.columns.drop(["Gender"
]), columns=["Coefficient"])

predict_label = lr.predict(X_test)

# ----- CONFUSION MATRIX ----- #

c_matrix = confusion_matrix(y_test, predict_label)

tp = c_matrix[1][1]
fp = c_matrix[0][1]
tn = c_matrix[0][0]
fn = c_matrix[1][0]

accuracy = (tp + tn) / len(predict_label)
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}".format(1-accuracy))
```

```
The accuracy is: 0.979381443298969
The error is: 0.020618556701030966
```

In [14]:
```
coeff_df
```

Out[14]:

	Coefficient
Race	-0.422005
Mental Health Issues	-0.020670
Total victims	0.130133

PROBLEM 2

In [15]:
```
# FEATURE PROBLEM 2 - predicting whether the shooter had mental health i
ssues
# This will be considered a Classification task. Classify/predict whethe
r the shooter was one or the other (binary)
```

In [16]:
```
# -- preprocessing / scaling (cont.) -- #

# create a copy of the project df for this specific problem
problem2_df = project_df

# get the indicies where the Mental Health Issues were unknown
indicies = problem2_df[problem2_df['Mental Health Issues'] == 2].index

# dropping values where Mental Health Issues == 2 (the Mental Health Iss
ues were unknown)
problem2_df = problem2_df.drop(indicies)

# reset the indicies so we don't get NaN values
problem2_df = problem2_df.reset_index(drop=True)

# transform the training and testing data to scale. drop non-numerical c
olumns
scaled_data = scaler.fit_transform(problem2_df.drop(['Mental Health Issu
es'], axis = 1))

# turn transformed/scaled data back into dataframes with header columns
scaled_df = pd.DataFrame(scaled_data, columns=problem2_df.columns.drop([
'Mental Health Issues']))
scaled_df['Mental Health Issues'] = problem2_df['Mental Health Issues']

# assign the features and target to the approriate columns/variables
features = scaled_df.drop(['Mental Health Issues'], axis = 1)
target = scaled_df['Mental Health Issues']
```

In [17]:
```
# For context, the data contains about 53% shooters with mental health i
ssues (106/199) and 47% shooters that don't
problem2_df['Mental Health Issues'].value_counts()
```

Out[17]:
```
1    106
0     93
Name: Mental Health Issues, dtype: int64
```

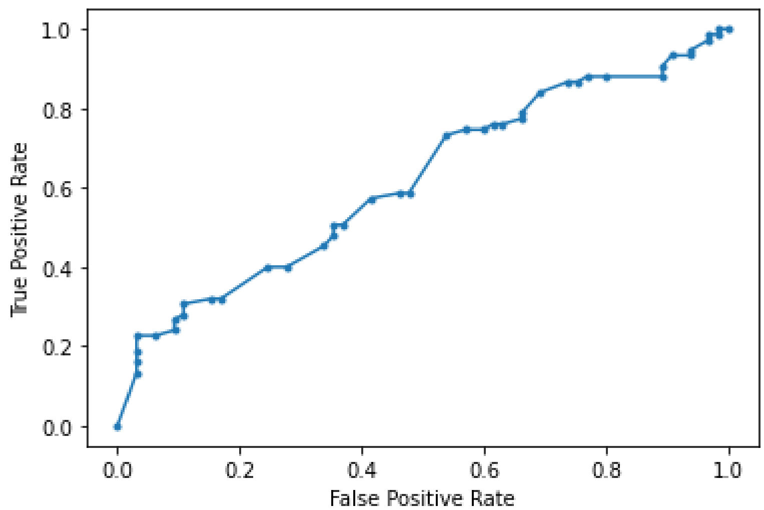
In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(features, target, te
st_size=0.7)

# scale the features
X_train = StandardScaler().fit_transform(X_train.values)
X_test = StandardScaler().fit_transform(X_test.values)
```

Model 1: Naive Bayes

In [19]:
```
# ----- MODEL 1: Naive Bayes ----- #

print("\n----- Naive Bayes -----")

nb = GaussianNB()
nb.fit(X_train, y_train)

# TRAINING
print("On Training Data:")
train_predict_label = nb.predict(X_train)

c_matrix = confusion_matrix(y_train, train_predict_label)

tp = c_matrix[1][1]
tn = c_matrix[0][0]

accuracy = (tp + tn) / (len(train_predict_label))
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}\n".format(1-accuracy))

# TESTING
print("On Testing Data:")
test_predict_label = nb.predict(X_test)

c_matrix = confusion_matrix(y_test, test_predict_label)

tp = c_matrix[1][1]
tn = c_matrix[0][0]

accuracy = (tp + tn) / len(test_predict_label)
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}\n".format(1-accuracy))

# ROC and AUC

probabilities = nb.predict_proba(X_test)
probabilities = probabilities[:, 1]
r_auc = roc_auc_score(y_test, probabilities)
print("AUC=", r_auc)

fpr, tpr, _ = roc_curve(y_test, probabilities)
plt.plot(fpr, tpr, marker='.', label="naive bayes")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
----- Naive Bayes -----
On Training Data:

The accuracy is: 0.6271186440677966
The error is: 0.3728813559322034

On Testing Data:

The accuracy is: 0.5714285714285714
The error is: 0.4285714285714286

AUC= 0.6174358974358973
```

Out[19]: `Text(0, 0.5, 'True Positive Rate')`



Model 2: Logistic Regression

In [20]:
```
# ----- MODEL 2: Logistic Regression ----- #

lr = LogisticRegression()
lr.fit(X_train, y_train)
coeff_df = pd.DataFrame(lr.coef_.T, problem1_df.columns.drop(["Mental He
alth Issues"]), columns=["Coefficient"])

predict_label = lr.predict(X_test)

# ----- CONFUSION MATRIX ----- #

c_matrix = confusion_matrix(y_test, predict_label)

tp = c_matrix[1][1]
fp = c_matrix[0][1]
tn = c_matrix[0][0]
fn = c_matrix[1][0]

accuracy = (tp + tn) / len(predict_label)
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}".format(1-accuracy))

# avoid a division by 0 error
if tp+fp > 0:
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
else:
    precision = 0
    recall = 0

# avoid a division by 0 error
if precision+recall > 0:
    f1 = 2 * (precision*recall) / (precision+recall)
else:
    f1 = 0

print("\nThe precision is: {}".format(precision))
print("The recall is: {}".format(recall))
print("The F1 score is: {}".format(f1))
```

```
The accuracy is: 0.5714285714285714
The error is: 0.4285714285714286

The precision is: 0.6363636363636364
The recall is: 0.4666666666666667
The F1 score is: 0.5384615384615385
```

In [21]:
```
coeff_df
```

Out[21]:

	Coefficient
Race	-0.005854
Gender	-0.370167
Total victims	0.908748

Model 3: kNearest Neighbors

In [22]:
```
# ----- MODEL 3: kNN ----- #

print("----- kNN -----")

ks = [2, 3, 4, 5, 6, 7, 8, 9, 10]
max_accuracy = 0
max_k = 0

for k in ks:

    print("k is: {}".format(k))

    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)

    predict_label = knn.predict(X_test)

    c_matrix = confusion_matrix(y_test, predict_label)

    tp = c_matrix[1][1]
    fp = c_matrix[0][1]
    tn = c_matrix[0][0]
    fn = c_matrix[1][0]

    accuracy = (tp + tn) / len(predict_label)

    if accuracy > max_accuracy:
        max_accuracy = accuracy
        max_k = k

    print("The accuracy is: {}".format(accuracy))
    print("The error is: {}\n".format(1-accuracy))

print("The k with the highest accuracy is {} with an accuracy of {}".for
mat(max_k, max_accuracy))
```

```
----- kNN -----
k is: 2
The accuracy is: 0.5142857142857142
The error is: 0.48571428571428577

k is: 3
The accuracy is: 0.5642857142857143
The error is: 0.4357142857142857

k is: 4
The accuracy is: 0.5285714285714286
The error is: 0.4714285714285714

k is: 5
The accuracy is: 0.55
The error is: 0.44999999999999996

k is: 6
The accuracy is: 0.5285714285714286
The error is: 0.4714285714285714

k is: 7
The accuracy is: 0.5285714285714286
The error is: 0.4714285714285714

k is: 8
The accuracy is: 0.5214285714285715
The error is: 0.47857142857142854

k is: 9
The accuracy is: 0.55
The error is: 0.44999999999999996

k is: 10
The accuracy is: 0.5857142857142857
The error is: 0.41428571428571426

The k with the highest accuracy is 10 with an accuracy of 0.58571428571
42857
```

PROBLEM 3

In [23]:

```
# FEATURE PROBLEM 3 - predicting the race of the shooter
# This will be considered a Classification task. Classify/predict the cl
ass of the shooter (class = race)
```

In [24]:
```
# -- preprocessing / scaling (cont.) -- #

# create a copy of the project df for this specific problem
problem3_df = project_df

# get the indicies where the Mental Health Issues were unknown
indicies = problem3_df[problem3_df['Race'] == 6].index

# dropping values where Mental Health Issues == 2 (the Mental Health Iss
ues were unknown)
problem3_df = problem3_df.drop(indicies)

# reset the indicies so we don't get NaN values
problem3_df = problem3_df.reset_index(drop=True)

# transform the training and testing data to scale. drop non-numerical c
olumns
scaled_data = scaler.fit_transform(problem3_df.drop(['Race'], axis = 1))

# turn transformed/scaled data back into dataframes with header columns
scaled_df = pd.DataFrame(scaled_data, columns=problem3_df.columns.drop([
'Race']))
scaled_df['Race'] = problem3_df['Race']

# assign the features and target to the approriate columns/variables
features = scaled_df.drop(['Race'], axis = 1)
target = scaled_df['Race']
```

In [25]:
```
# For context, the data contains about:
# 56% of the shooters were white (144/257)
# 33% of the shooters were black (85/257)
# 7% of the shooters were asian (18/257)
# 3% of the shooters were native american (8/257)
# <1% of the shooters were 2 or more races (2/257)
problem3_df['Race'].value_counts()
```

Out[25]:
```
0    144
1     85
2     18
4      8
5      2
Name: Race, dtype: int64
```

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(features, target, te
st_size=0.5)

# scale the features
X_train = StandardScaler().fit_transform(X_train.values)
X_test = StandardScaler().fit_transform(X_test.values)
```

Model 1: kNearest Neighbors

In [27]:
```
# ----- MODEL 1: kNN ----- #

print("----- kNN -----")

ks = [2, 3, 4, 5, 6, 7, 8, 9, 10, 50]
max_accuracy = 0
max_k = 0

for k in ks:

    print("k is: {}".format(k))

    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)

    predict_label = knn.predict(X_test)

    c_matrix = confusion_matrix(y_test, predict_label)

    tp = c_matrix[1][1]
    fp = c_matrix[0][1]
    tn = c_matrix[0][0]
    fn = c_matrix[1][0]

    accuracy = (tp + tn) / len(predict_label)

    if accuracy > max_accuracy:
        max_accuracy = accuracy
        max_k = k

    print("The accuracy is: {}".format(accuracy))
    print("The error is: {}\n".format(1-accuracy))

print("The k with the highest accuracy is {} with an accuracy of {}".for
mat(max_k, max_accuracy))
```

```
----- kNN -----
k is: 2
The accuracy is: 0.5891472868217055
The error is: 0.4108527131782945

k is: 3
The accuracy is: 0.5193798449612403
The error is: 0.48062015503875966

k is: 4
The accuracy is: 0.5348837209302325
The error is: 0.4651162790697675

k is: 5
The accuracy is: 0.5193798449612403
The error is: 0.48062015503875966

k is: 6
The accuracy is: 0.5038759689922481
The error is: 0.49612403100775193

k is: 7
The accuracy is: 0.5038759689922481
The error is: 0.49612403100775193

k is: 8
The accuracy is: 0.5193798449612403
The error is: 0.48062015503875966

k is: 9
The accuracy is: 0.5193798449612403
The error is: 0.48062015503875966

k is: 10
The accuracy is: 0.5271317829457365
The error is: 0.4728682170542635

k is: 50
The accuracy is: 0.5658914728682171
The error is: 0.43410852713178294

The k with the highest accuracy is 2 with an accuracy of 0.589147286821
7055
```

Model 2: Support Vector Machine

In [28]:
```
# ----- MODEL 2: SVM ----- #

svm = SVC()
svm.fit(X_train, y_train)
predict_label = svm.predict(X_test)

c_matrix = confusion_matrix(y_test, predict_label)

tp = c_matrix[1][1]
fp = c_matrix[0][1]
tn = c_matrix[0][0]
fn = c_matrix[1][0]

accuracy = (tp + tn) / len(predict_label)

if accuracy > max_accuracy:
    max_accuracy = accuracy
    max_k = k

print("The accuracy is: {}".format(accuracy))
print("The error is: {}".format(1-accuracy))
```

```
The accuracy is: 0.5193798449612403
The error is: 0.48062015503875966
```

PROBLEM 4

In [29]:
```
# FEATURE PROBLEM 4 - what is considered a mass shooting?
# This will be considered a Regression task.
# Predicting the # of victims of a mass shooting
```

In [30]:
```
# -- preprocessing / scaling (cont.) -- #

# create a copy of the project df for this specific problem
problem4_df = project_df

# transform the training and testing data to scale. drop non-numerical c
olumns
scaled_data = scaler.fit_transform(problem4_df.drop(['Total victims'], a
xis = 1))

# turn transformed/scaled data back into dataframes with header columns
scaled_df = pd.DataFrame(scaled_data, columns=problem4_df.columns.drop([
'Total victims']))
scaled_df['Total victims'] = problem4_df['Total victims']

# assign the features and target to the approriate columns/variables
features = scaled_df.drop(['Total victims'], axis = 1)
target = scaled_df['Total victims']
```

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split(features, target, te
st_size=0.5)

# scale the features
X_train = StandardScaler().fit_transform(X_train.values)
X_test = StandardScaler().fit_transform(X_test.values)
```

Model 1: Linear Regression

In [32]:
```
# ----- MODEL 1: Linear Regression ----- #

mlr = LinearRegression()
mlr.fit(X_train, y_train)

# retrieve the coefficients of the data
coeff_df = pd.DataFrame(mlr.coef_, problem4_df.columns.drop(["Total vict
ims"]), columns=["Coefficient"])

y_train_predict = mlr.predict(X_train)
mse = mean_squared_error(y_train, y_train_predict)

print("On Training Data:")
print("\nThe MSE is {}".format(mse))
print("The RMSE is {}\n".format(math.sqrt(mse)))

y_test_predict = mlr.predict(X_test)
mse = mean_squared_error(y_test, y_test_predict)

print("On Testing Data:")
print("\nThe MSE is {}".format(mse))
print("The RMSE is {}\n".format(math.sqrt(mse)))
```

```
On Training Data:

The MSE is 2119.2777353679658
The RMSE is 46.035613772034864

On Testing Data:

The MSE is 157.0660073900018
The RMSE is 12.53259779096105
```

In [33]:
```
coeff_df
```

Out[33]:

	Coefficient
Race	-5.410165
Gender	-0.005663
Mental Health Issues	4.797411

MORE DATA PROCESSING

More processing of data for regression tasks, to transform them into Classifcation tasks, and so we can work with more variables

In [34]:
```
# For this task, I will encode the following:
# - Total victims as a variable where:
# 0 = 0-9 victims
# 1 = 10-19 victims
# 2 = 20-29 victims
# 3 = 30-39 victims
# 4 = 40-49 victims
# 5 = 50-59 victims
# 6 = 60-69 victims
# 7 = 70-79 victims
# 8 = 80-89 victims
# 9 = 90-99 victims
# 10 = 100+ victims
# -------------------------------------------------------

totals_df = []


# ----- (1) ENCODING VICTIMS AS A VARIABLE -----

for i in range(len(data)):
    totals_index = math.floor(data['Total victims'][i] / 10)

    if totals_index >= 10:
        totals_df.append(10)

    else:
        totals_df.append(totals_index)

project_df["Total victims"] = totals_df
```

In [35]:
```
project_df
```

Out[35]:

	Race	Gender	Mental Health Issues	Total victims
0	0	0	0	4
1	0	0	0	0
2	1	0	0	0
3	0	0	2	10
4	2	0	1	0
...	...	...	...	...
318	6	0	1	0
319	1	0	1	2
320	0	0	1	0
321	0	0	1	0
322	0	0	1	4

323 rows × 4 columns

In [36]:
```
# FEATURE PROBLEM 4 (PART 2) - predicting how many victims of a mass sho
oting
# This will (NOW) be considered a CLASSIFICATION task.
# Classifying/predicting the class for which the # of victims of a mass
 shooting will belong (class = bucket)
```

In [37]:
```
# -- preprocessing / scaling (cont.) -- #

# create a copy of the project df for this specific problem
problem4_df = project_df

# transform the training and testing data to scale. drop non-numerical c
olumns
scaled_data = scaler.fit_transform(problem4_df.drop(['Total victims'], a
xis = 1))

# turn transformed/scaled data back into dataframes with header columns
scaled_df = pd.DataFrame(scaled_data, columns=problem4_df.columns.drop([
'Total victims']))
scaled_df['Total victims'] = problem4_df['Total victims']

# assign the features and target to the approriate columns/variables
features = scaled_df.drop(['Total victims'], axis = 1)
target = scaled_df['Total victims']
```

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(features, target, te
st_size=0.9)

# scale the features
X_train = StandardScaler().fit_transform(X_train.values)
X_test = StandardScaler().fit_transform(X_test.values)
```

Model 1: Logistic Regression

In [39]:
```
# ----- MODEL 2: Logistic Regression ----- #

lr = LogisticRegression()
lr.fit(X_train, y_train)

# retrieve the coefficients of the data
predict_label = lr.predict(X_test)

# ----- CONFUSION MATRIX ----- #

c_matrix = confusion_matrix(y_test, predict_label)

tp = c_matrix[1][1]
fp = c_matrix[0][1]
tn = c_matrix[0][0]
fn = c_matrix[1][0]

accuracy = (tp + tn) / len(predict_label)
print("\nThe accuracy is: {}".format(accuracy))
print("The error is: {}".format(1-accuracy))

# avoid a division by 0 error
if tp+fp > 0:
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
else:
    precision = 0
    recall = 0

# avoid a division by 0 error
if precision+recall > 0:
    f1 = 2 * (precision*recall) / (precision+recall)
else:
    f1 = 0

print("\nThe precision is: {}".format(precision))
print("The recall is: {}".format(recall))
print("The F1 score is: {}".format(f1))
```

```
The accuracy is: 0.7560137457044673
The error is: 0.24398625429553267

The precision is: 0
The recall is: 0
The F1 score is: 0
```