```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import warnings
import math
from copy import copy

#Preprocessing
from sklearn.preprocessing import StandardScaler, normalize, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline

#Classification Models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC

import sklearn.metrics, sklearn.model_selection, sklearn.tree
import sklearn.neural_network


from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR


# Accuracy Metrics
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.metrics import accuracy_score, precision_score, recall_score, mean_


import plotly.graph_objs as go
import plotly.express as px

warnings.filterwarnings('ignore')
```

```
#This is the dataset for the Death In Custody
#In order to let both users collaborate on the notebook we each need to apply ou

#Dylan's Path
#deathCustody = pd.read_excel('/content/drive/MyDrive/Grad School/BDA 600/BDA 60

#Layth's Path
deathCustody = pd.read_excel(open('/content/drive/MyDrive/BDA 600 - Project/Deat
#deathCustody = deathCustody.dropna() #Remove all NaN valued data points for bet

deathCustody.isna().sum()
```

```
record_key_number                           0
reporting_agency                            0
agency_number                               0
agency_full_name                            0
county                                      0
latitude                                    0
longitude                                   0
race                                        2
gender                                      0
age                                         0
custody_status                              0
custody_offense                             0
date_of_death_yyyy                          0
date_of_death_mm                            0
date_of_death_dd                            0
custodial_responsibilty_at_time_of_death    0
location_where_cause_of_death_occurred      0
facility_death_occured                      1
manner_of_death                             0
means_of_death                              0
dtype: int64
```

The Data

This dataset contains information on deaths related to police custody in California form 2005 and 2021. Each row represents a death and includes basic demographic information (age, race, gender) as well as some other features including: location, custodial responsibilty, custody status and the offence the criminal commited.

The first step we took was to deal with the NA values. Thankfully this data set only had 3 NA values which represented a small portion of the 12k+ dataset. Two rows didnt have the "race" value filled in so we classified this as "other". The other NA was for "facility_death_occured" so we matched that with the "custodial_responsibilty_at_time_of_death". These are the first two assumttions we made for this dataset.

```
deathCustody.loc[deathCustody["race"].isna(),"race"] = "Other" # Changing NA val
deathCustody.loc[deathCustody["facility_death_occured"].isna(),"facility_death_o
deathCustody.isna().sum()
```

```
record_key_number                           0
reporting_agency                            0
agency_number                               0
agency_full_name                            0
county                                      0
latitude                                    0
longitude                                   0
race                                        0
gender                                      0
age                                         0
custody_status                              0
custody_offense                             0
date_of_death_yyyy                          0
date_of_death_mm                            0
date_of_death_dd                            0
custodial_responsibilty_at_time_of_death    0
location_where_cause_of_death_occurred      0
facility_death_occured                      0
manner_of_death                             0
means_of_death                              0
dtype: int64
```

We also included a dataset that has some information of various statistis of each zipcode in California in the year 2021. We chose the columns that we surmised would be the most correlated with the death data. Our second assumption for this data is that these statistis were similar in 2021 as they were for the 6 previous years. In order to merge the two datasets, we had to average the metrics of all the zipcodes in the same county to be able to match up to the deaths in custody.

```python
# Loading it the 2021 California Environment Metric Data

calenvi_cols = ['Census Tract',
                'California County',
                'Total Population',
                ' CES 4.0 Percentile',
                "Ozone Pctl",
                'Tox. Release',
                'Solid Waste',
                'Pollution Burden',
                'Education',
                'Linguistic Isolation',
                'Poverty',
                'Unemployment',
                'Housing Burden']
#Layth's Path
calEnvi = pd.read_excel('/content/drive/MyDrive/BDA 600 - Project/calenviroscree

#Dylan's Path
#calEnvi = pd.read_excel('/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Pro
calEnvi.head(5)
```

	Census Tract	Total Population	California County	CES 4.0 Percentile	Ozone Pctl	Tox. Release	Solid Waste	Po
0	6001981900	58	Alameda	NaN	3.123833	466.708995	0.0	4
1	6001982000	63	Alameda	NaN	3.123833	461.068003	0.0	5
2	6001422600	1043	Alameda	NaN	3.123833	694.326325	0.0	3
3	6009000504	457	Calaveras	NaN	68.176727	1.598387	6.2	1
4	6019007901	3251	Fresno	NaN	63.609210	284.350006	0.0	2

```
# Loading it the 2021 California Demographic Metric Data
caldemo_cols = ['Census Tract',
                'Children < 10 years (%)',
                'Pop 10-64 years (%)',
                'Elderly > 64 years (%)',
                'Hispanic (%)',
                'White (%)',
                'African American (%)',
                'Native American (%)',
                'Asian American (%)',
                'Other/Multiple (%)']
#Layth's Path
calDemo = pd.read_excel('/content/drive/MyDrive/BDA 600 - Project/calenviroscree

#Dylan's Path
#calDemo = pd.read_excel('/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Pro
calDemo.head(5)
```

```
                Children    Pop  Elderly                        African   Native
      Census        < 10  10-64     > 64  Hispanic  White      American American  Ame
       Tract       years  years    years       (%)    (%)           (%)      (%)
                     (%)    (%)      (%)
0 6001400100        7.82  66.12    26.06      3.78  74.26          3.43     0.00
1 6001400200       10.46  66.32    23.22      8.67  73.49          2.59     0.20
2 6001400300       11.42  73.04    15.54      6.95  67.99          9.09     0.00
3 6001400400        9.38  78.79    11.83     12.10  63.74          6.64     0.87
4 6001400500        9.12  81.96     8.92      9.46  45.44         21.39     0.00
```

Now that we have called in the two sheets of the 2021 California Enviromental Screening Dataset, we need to merge them together and average their values for each of the counties to get them into a format we can include them into our DeathCustody dataset that we will be using for our machine learning models later on.

```
calMetrics = calEnvi.merge(calDemo, on = 'Census Tract',how = "left")
calMetrics["California County"] = calMetrics["California County"].str.strip()

calMetrics_avgs = calMetrics.drop("Census Tract",axis = 1).groupby("California C
calMetrics_avgs.reset_index(inplace = True)
calMetrics_avgs.head()

```

	California County	Total Population	CES 4.0 Percentile	Ozone Pctl	Tox. Release	Solid Waste	Pollution Burden
0	Alameda	4602.094444	41.310417	10.875786	466.776452	1.004444	37.323623
1	Alpine	1039.000000	21.999496	62.526447	0.596264	8.000000	33.241256
2	Amador	4269.888889	38.769821	66.977805	36.165427	8.788889	36.674443
3	Butte	4427.784314	39.791166	61.685030	1.385422	2.635294	34.389987
4	Calaveras	4551.400000	27.954278	71.717486	20.067040	11.475000	28.820814

5 rows × 21 columns

Below we are checking to see which of the counties are matching betweent the datasets. There are 56 matching counties and each have two that the other doesnt have. We can disregard the extra Counties from the calMetrics dataset since our Death Custody is the dataset we will be running our machine learning models on. The other two "counties" in the death dataset are 'In-State' and 'Out-of-State'. These variables make up 153 rows so we decided to drop them since we dont have CalMetric information for these. After dropping these rows we are left with 12329 rows and 41 columns. We will continue to do some more preprocessing on our data later on to further prepare it for our models.

```python
calmetric_set = set(calMetrics_avgs["California County"])
death_set = set(deathCustody["county"].str.strip())
print(len(calmetric_set))
print(len(death_set))
print(calmetric_set.symmetric_difference(death_set))
```

```
58
58
{'Alpine', 'Out-of-State', 'In-State', 'Plumas'}
```

```python
deathCustody2 = deathCustody[deathCustody["county"].isin(["Out-of-State",'In-Sta
calMetrics_avgs2 = calMetrics_avgs[calMetrics_avgs["California County"].isin(['A

merged_data = deathCustody2.merge(calMetrics_avgs2,left_on = "county", right_on

merged_data['date_of_death'] = pd.to_datetime(merged_data['date_of_death_yyyy'].
                                    merged_data['date_of_death_mm'].astype(str)
                                    merged_data['date_of_death_dd'].astype(str)

# Drop the original date columns
merged_data.drop(['date_of_death_yyyy', 'date_of_death_mm', 'date_of_death_dd'],

# Convert the combined date column to datetime type
merged_data['date_of_death'] = pd.to_datetime(merged_data['date_of_death'])

# Verify the combined date column and its data type
merged_data.shape
```

```
(12329, 39)
```

Manipulating the Dataset

Here we wanted in any case of combining this dataset of Death Custody to the other additional dataset with social and enviornmental levels of the population together to produce greater insight and analysis. To accomplish this a momentary function to insert the Longitude and Latitude into the Death Custody dataset and then save it to the CSV, to avoid the need to rerun the function every time.

There should be mention that we do comment out these code sections to allow us to run the rest of the code without running this section since it is essentailly a one time running process for manipulating the original datasheet.

[ ] ↳ *6 cells hidden*

Overview of our Data

Our data up to this point containes infomation from the Death in Custody datast, 18 columns detailing each death collected from 2005-2021. This was combined with 21 select columns from the California Enviro Screen excel sheets. Below we will give information on each column, what preprocesing steps that have been conducted and will be done.

Keep

Target variables:

- **RACE** -> This will be our nominal dependent variable that we will be using multinomial classifcaion models ot predict. We will sort this column into White, Hispanic, Black, Other. Other Contains all of the other race categories in the dataset since the first three races make up the majority of the dataset.
- **AGE** -> This will be our discrete dependent variable. We will run various regression models on the data to predict this column.

Each of these columns will be used for the prediction models of the other

Features:

- Record_key_number -> The primary key that came with the Deaths in custody dataset which we will turn into our index.
- Reporting_agency -> 'Other Local' value will be placed with 'Police' value and 'Probation' will be placed under the 'State' value. We will then turn this into integers from 0-2 since we can catorgieze this feature as ordenial. We will sort it by the area of their juristiction.
- date of death -> We made this column by combining the month day and year column into one variable. Could use to see if there is any seasonality in the data.
- gender -> will make 0 male and 1 female and rename to gender_female
- custody_status -> We will organize this column into 'sentenced' and 'not sentenced' rename to "sentenced" for changing this column into 0-1
- custody offence -> This column will sorted by severity of offences (fellonly, misdominear, and status offence), turning this into an ordinal feature we can than turn into integers.
- 'custodial_responsibilty_at_time_of_death' -> This column shows who had custody of the person at time of death, we will do some grouping here as well so we can get it down to 5 categoroies. This is a nominal column so we will use one hot encoding to prepare it for the machine learning models.
- 'facility_death_occured' -> This column shows where the person actaully died. We will do some grouping here so the categories will match with the custodial_responsibilty_at_time_of_death column. We will then make a new column to check if these match to see if that is predictive. This is a nominal column so we will use one hot encoding to prepare it for the machine learning models.
- 'means of death' -> This column containes information of what was used in the cause of death of the person. We grouped this column into 5 different categories. This is a nominal column so we will use one hot encoding to prepare it for the machine learning models.
- 'manner-of-death' -> This column containes information of the classificaion on the death (natural causes, Homicide, etc.). We grouped this column into 6 different categories. This is a nominal column so we will use one hot encoding to prepare it for the machine learning models.
- 'Total Population' -> 2019 ACS population estimates in census tracts.
- 'CES 4.0 Score', -> CalEnviroScreen Score, Pollution Score multiplied by Population

Characteristics Score

- ' CES 4.0 Percentile', -> percentile of the above score
- 'Tox. Release'-> Toxicity-weighted concentrations of modeled chemical releases to air from facility emissions and off-site incineration (from RSEI)
- 'Solid Waste' -> Sum of weighted solid waste sites and facilities (SWIS) within buffered distances to populated blocks of census tracts.
- 'Pollution Burden'-> Average of percentiles from the Pollution Burden indicators (with a half weighting for the Environmental Effects indicators).
- 'Education' -> Percent of population over 25 with less than a high school education.
- 'Linguistic Isolation' ->Percent limited English speaking household.
- 'Poverty' -> Percent of population living below two times the federal poverty level.
- 'Unemployment' -> Percent of the population over the age of 16 that is unemployed and eligible for the labor force.
- 'Housing Burden' ->Percent housing burdened low income households.

Drop

- Agency_full_name -> The full name of the agency that had custody at time of death. We dont need for the ml models since there are too many catergoies for it to be predictive for this size of a dataset.
- agency_number -> Same as above.
- latitude -> This is based off the county location so we dont need it.
- longitude -> This is based off the county location so we dont need it.
- 'location_where_cause_of_death_occurred' -> This column didnt make much sense so we didnt want to include it since that would lower the explainiably of the model.
- county -> We used this column to merge the California Environmental screening data sets into our main deathCustody data. We will likely remove later but will test the predictiveness using the models first.
- 'California County' -> this column was only used to merge the datasets.

[ ] ↳ *24 cells hidden*

Machine Learning Modeling:

Now that we have prepared our data, making sure all our features are numeric, we are ready to start implementing some machine learning models. We will be conducting both classificaion and regression modeling for this dataset since we have two columns that will work well as target variables.

Classfication Models:

We will be running multi class classificaion models on the Race column to see if we can predict the race of the decseaed based off our features. If we have good accuracy for these models, we will check which features are the most predictive to show if there are sytematic issues that can be addressed.
We will be testing each of the models below to see which has the best accuracy and which can tell us the most about the data using feature importance.

1. k-Nearest Neighbors.
2. Decision Trees.
3. Naive Bayes.
4. Random Forest.
5. Gradient Boosting.
6. Support Vector Classifier

[ ] ↳ *3 cells hidden*

Predicting by Age

[ ] ↳ *15 cells hidden*

Predicting by Race

```
mlcustody_final.columns
```

```
race_pred_data = mlcustody_final.drop(['race_Black', 'race_Hispanic', 'race_Othe
       'date_of_death_dd', 'Total Population', ' CES 4.0 Percentile',
       'Ozone Pctl', 'Tox. Release', 'Solid Waste', 'Pollution Burden',
       'Education', 'Linguistic Isolation', 'Poverty', 'Unemployment',
       'Housing Burden', 'Children < 10 years (%)', 'Pop 10-64 years (%)',
       'Elderly > 64 years (%)', 'Hispanic (%)', 'White (%)',
       'African American (%)', 'Native American (%)', 'Asian American (%)',
       'Other/Multiple (%)', 'custodial_responsibilty_at_time_of_death_CDC/CRC',
       'custodial_responsibilty_at_time_of_death_Crime Scene',
       'custodial_responsibilty_at_time_of_death_Hospital',
       'custodial_responsibilty_at_time_of_death_Jail',
       'custodial_responsibilty_at_time_of_death_Other',
       'facility_death_occured_CDC/CRC', 'facility_death_occured_Crime Scene',
       'facility_death_occured_Hospital', 'facility_death_occured_Jail',
       'facility_death_occured_Other', 'manner_of_death_Accidental',
       'manner_of_death_Homicide Justified (Law Enforcement Staff & Other Inmate
       'manner_of_death_Homicide Willful (Law Enforcement Staff & Other Inmate)'
       'manner_of_death_Natural', 'manner_of_death_Other',
       'manner_of_death_Suicide', 'means_of_death_Drug Overdose',
       'means_of_death_Hanging,Strangulation', 'means_of_death_Not Applicable',
       'means_of_death_Other', 'means_of_death_Weapons of All Manner',], axis =1

X = race_pred_data.loc[:, race_pred_data.columns != "race"]


y = race_pred_data.loc[:, race_pred_data.columns == "race"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

print(X_train.columns)
print(y_train["race"].value_counts())
```

```
knn = KNeighborsClassifier(n_neighbors = 3 )

knn_preds = knn.fit(X_train,y_train).predict(X_test)

knn_cv_score_a = cross_val_score(knn, X, y, cv=10, scoring = "accuracy" )
# knn_cv_score_p = cross_val_score(knn, X, y, cv=10, scoring = "precision" )
# knn_cv_score_r = cross_val_score(knn, X, y, cv=10, scoring = "recall" )
# knn_cv_score_f = cross_val_score(knn, X, y, cv=10, scoring = "f1" )
# knn_p_score = precision_score(y_test,knn_preds, labels = )

# knn.fit(X_train,y_train)
# knn_score = knn.score(X_test,y_test)

print(np.mean(knn_cv_score_a))
# print(np.mean(knn_cv_score_p))
# print(np.mean(knn_cv_score_r))
# print(np.mean(knn_cv_score_f))

print(classification_report(y_test,knn_preds, target_names = ["Black","Hispanic"

conf_matrix = confusion_matrix( y_test,knn_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Black","Hispanic","O
disp.plot()
plt.show()
```

```
gnb = GaussianNB()

gnb_cv_score = cross_val_score(gnb, X, y, cv=10)

print(np.mean(gnb_cv_score))

gnb_preds = gnb.fit(X_train,y_train).predict(X_test)

print(classification_report(y_test,gnb_preds, target_names = ["Black","Hispanic"

conf_matrix = confusion_matrix(y_test,gnb_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Black","Hispanic","O
disp.plot()
plt.show()
```

```
rfc = RandomForestClassifier(max_depth=2, random_state=0)

rfc_cv_score = cross_val_score(rfc, X, y, cv=10)

rfc_preds = rfc.fit(X_train,y_train).predict(X_test)

print(classification_report(y_test,rfc_preds, target_names = ["Black","Hispanic"


print(np.mean(rfc_cv_score))

conf_matrix = confusion_matrix(y_test,rfc_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Black","Hispanic","C
disp.plot()
plt.show()
```

```
gbc = GradientBoostingClassifier(n_estimators=100,
                                 learning_rate=1.0,
                                 max_depth=1,
                                 max_features = "auto",
                                 random_state=0).fit(X_train, y_train)

gbc_cv_score = cross_val_score(gbc, X, y, cv=10)

print(np.mean(gbc_cv_score))

gbc_fit = gbc.fit(X_train,y_train)
gbc_preds =gbc_fit.predict(X_test)
gbc_proba = gbc_fit.predict_proba(X_test)


conf_matrix = confusion_matrix(y_test,gbc_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Black","Hispanic","C
disp.plot()
plt.show()
```

```
svc = make_pipeline(StandardScaler(), SVC(gamma='auto'))

svc_cv_score = cross_val_score(svc, X, y, cv=10)

print(np.mean(svc_cv_score))


svc_preds = svc.fit(X_train,y_train).predict(X_test)

conf_matrix = confusion_matrix(y_test,svc_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Black","Hispanic","C
disp.plot()
plt.show()
```

Predicting by Agency

```
agency_pred_data = mlcustody_final.drop([ 'date_of_death_yyyy', 'date_of_death_m
       'date_of_death_dd', 'Total Population', ' CES 4.0 Percentile',
       'Ozone Pctl', 'Tox. Release', 'Solid Waste', 'Pollution Burden',
       'Education', 'Linguistic Isolation', 'Poverty', 'Unemployment',
       'Housing Burden', 'Children < 10 years (%)', 'Pop 10-64 years (%)',
       'Elderly > 64 years (%)', 'Hispanic (%)', 'White (%)',
       'African American (%)', 'Native American (%)', 'Asian American (%)',
       'Other/Multiple (%)', 'sentenced'], axis =1 )

X = agency_pred_data.loc[:, agency_pred_data.columns != "reporting_agency"]


y = agency_pred_data.loc[:, agency_pred_data.columns == "reporting_agency"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

```
knn = KNeighborsClassifier(n_neighbors = 3 )

knn_preds = knn.fit(X_train,y_train).predict(X_test)

knn_cv_score = cross_val_score(knn, X, y, cv=10)

# knn.fit(X_train,y_train)
# knn_score = knn.score(X_test,y_test)

print(np.mean(knn_cv_score))
conf_matrix = confusion_matrix(y_test,knn_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Local Police", "Sher
disp.plot()
plt.show()
```

```
gnb = GaussianNB()

gnb_cv_score = cross_val_score(gnb, X, y, cv=10)

print(np.mean(gnb_cv_score))

gnb_preds = gnb.fit(X_train,y_train).predict(X_test)


conf_matrix = confusion_matrix(y_test,gnb_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Local Police", "Sher
disp.plot()
plt.show()
```

```
gbc = GradientBoostingClassifier(n_estimators=100,
                                 learning_rate=1.0,
                                 max_depth=1,
                                 max_features = "auto",
                                 random_state=0).fit(X_train, y_train)

#gbc_cv_score = cross_val_score(gbc, X, y, cv=10)



print(np.mean(gbc_cv_score))

gbc_preds = gbc.fit(X_train,y_train).predict(X_test)

print(classification_report(y_test,gbc_preds, target_names = ["Local Police", "S

conf_matrix = confusion_matrix(y_test,gbc_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Local Police", "Sher
disp.plot()
plt.show()
```

```
rfc = RandomForestClassifier(max_depth=2, random_state=0)

rfc_cv_score = cross_val_score(rfc, X, y, cv=10)

rfc_preds = rfc.fit(X_train,y_train).predict(X_test)



print(np.mean(rfc_cv_score))

conf_matrix = confusion_matrix(y_test,rfc_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Local Police", "Sher
disp.plot()
plt.show()
```

```
svc = make_pipeline(StandardScaler(), SVC(gamma='auto'))

svc_cv_score = cross_val_score(svc, X, y, cv=10)

print(np.mean(svc_cv_score))


svc_fit = svc.fit(X_train,y_train)

svc_preds = svc_fit.predict(X_test)
svc_proba = svc_fit.predict_proba(X_test)

conf_matrix = confusion_matrix(y_test,svc_preds)
print(conf_matrix)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=["Local Police", "Sher
disp.plot()
plt.show()
```

Exploring More of the Data:

Now that we have finished most of what we want to do with our data including manipulation for ML Models. Here we looked into making a new data frame that looks at the death rate of each county per 100,000 people. The data is normalized and scaled. This new infromation uses the full populatio count of each county from the CalEnviScreen data set and the total deaths per a county from the DeathCustody dataset.

[ ] ↳ *21 cells hidden*

Visualizations to create:

- Deaths normalized by the race %
- deaths per county (normalized)
- deaths per agency

time series:

- deaths per agency over time
- deaths in each race over time (normailzed?)

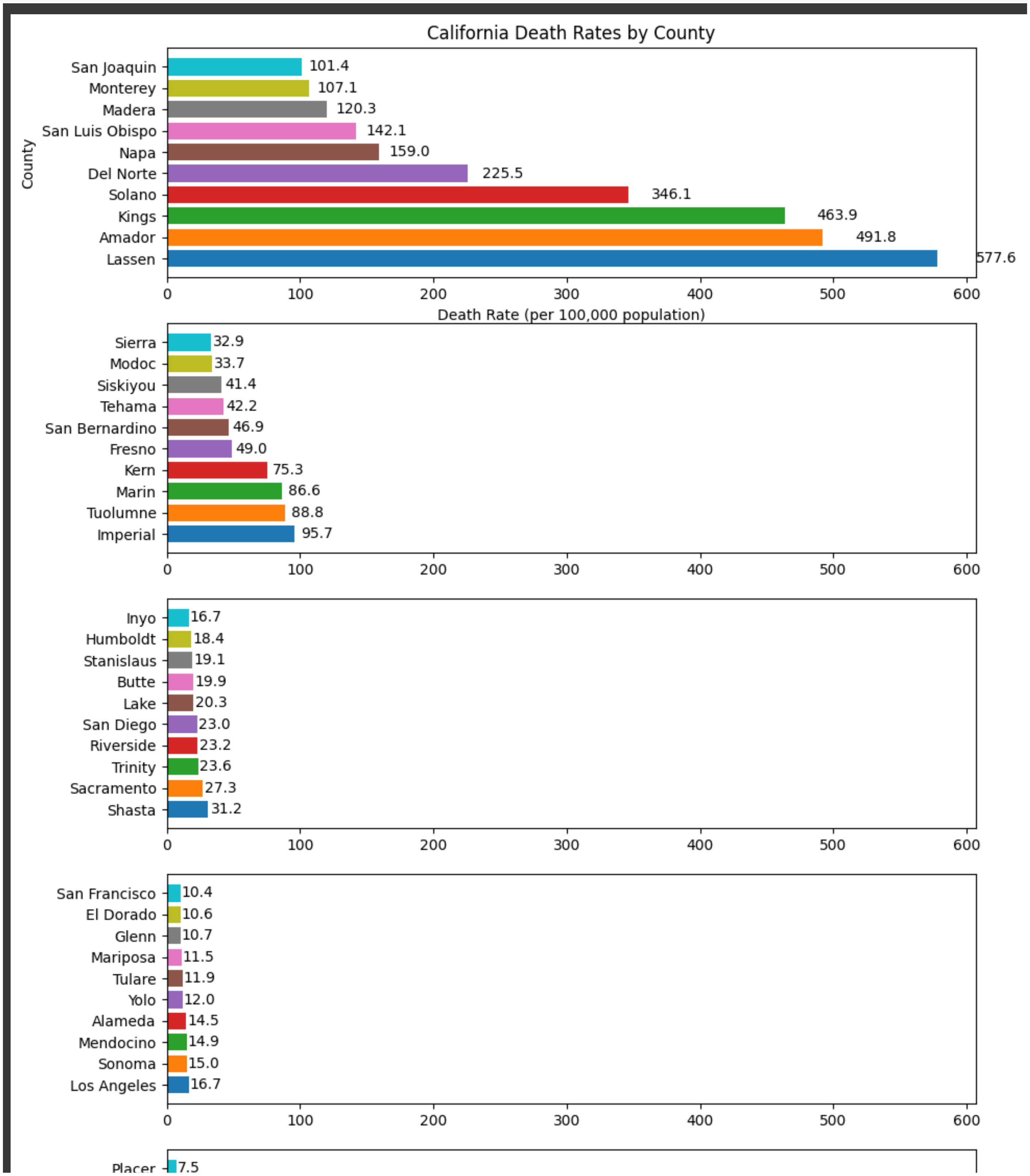
Data Visualization

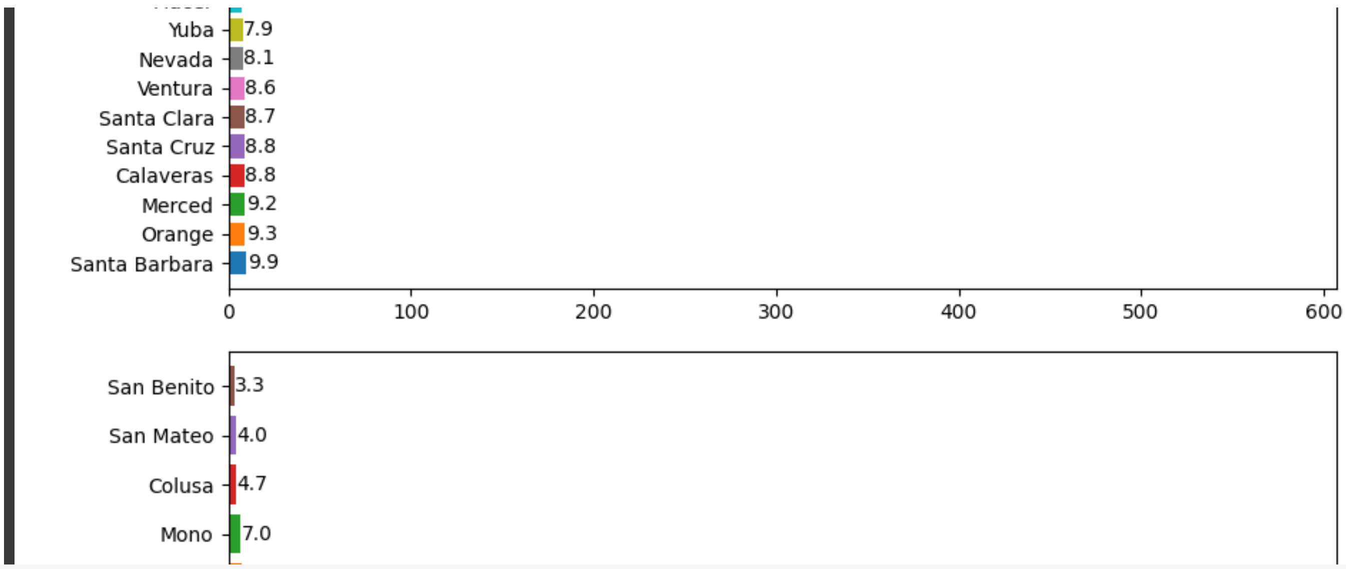
We will be looking into creating different visualizations for the newly manipulated and cleaned datasets we have. Looking at relations between different attributes of Death in Custody and Measurements of Californian Social and Economical reportings (i.e. Poverty, unemployment levels and ethnicity and gender).

Data viz ideas:

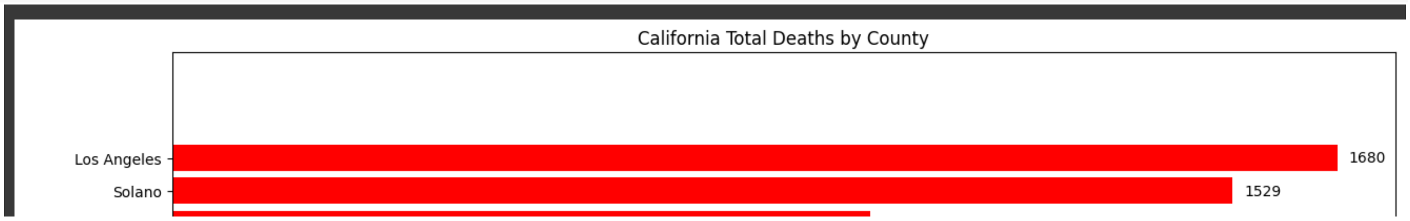
- show race levels for each county and then the death levels for each of those races to see if there is any stand out counties based off their percentage population.

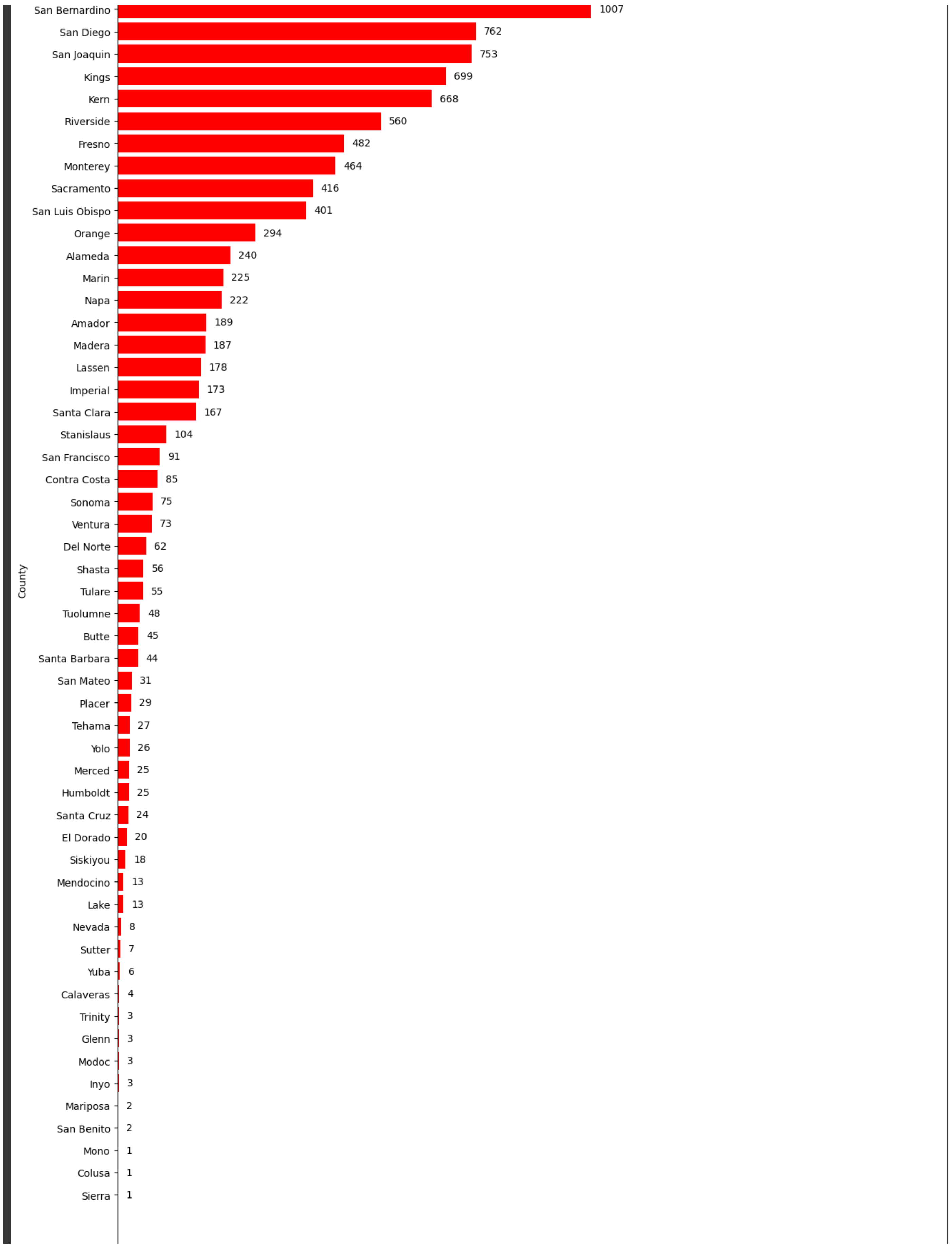
```
# Sort counties by death rate
mergedCounty = mergedCounty.sort_values(by='DeathRate', ascending=False)

# Create multiple subplots for each group of 10 counties
fig, axs = plt.subplots(nrows=math.ceil(len(mergedCounty)/10), ncols=1, figsize=
# Set axis labels and title for the last subplot
axs[0].set_xlabel('Death Rate (per 100,000 population)')
axs[0].set_ylabel('County')
axs[0].set_title('California Death Rates by County')
# Loop through the data and create a horizontal bar chart for each group of 10 c
for i, (index, row) in enumerate(mergedCounty.iterrows()):
    ax = axs[i//10]
    ax.barh(index, row['DeathRate'])
    ax.set_xlim(0, mergedCounty['DeathRate'].max()*1.05)

    # Add the death rate value next to the bar
    ax.text(row['DeathRate']*1.05, index, f"{row['DeathRate']:.1f}", va='center'

plt.savefig('CaliforniaDeathRatesCounty.png')
plt.show()
```

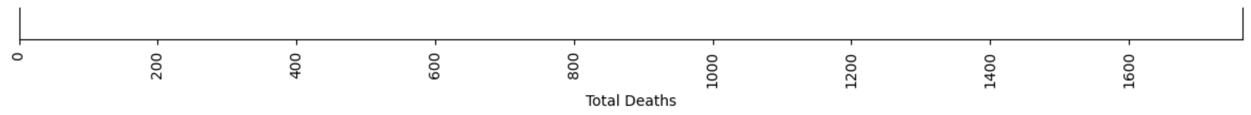




```
#Visualize the total number of deaths in each county. This is for comparing the
# Sort counties by death rate
mergedCounty = mergedCounty.sort_values(by='Deaths', ascending=True)

# Create a horizontal bar chart
fig, ax = plt.subplots(figsize=(15,25))
bars = ax.barh(mergedCounty.index, mergedCounty['Deaths'], color='red')

# Add text labels on top of each bar
for bar in bars:
    value = bar.get_width()
    x_pos = value + mergedCounty['Deaths'].max() * 0.01
    y_pos = bar.get_y() + bar.get_height() / 2
    ax.text(x_pos, y_pos, f'{value:.0f}', va='center', ha='left')

# Set axis labels and title
ax.set_xlabel('Total Deaths')
ax.set_ylabel('County')
ax.set_title('California Total Deaths by County')

# Rotate x-axis labels
plt.xticks(rotation=90)

plt.savefig('CaliforniaTotalDeathsCounty.png')
plt.show()
```




County
San Bernardino 1007
San Diego 762
San Joaquin 753
Kings 699
Kern 668
Riverside 560
Fresno 482
Monterey 464
Sacramento 416
San Luis Obispo 401
Orange 294
Alameda 240
Marin 225
Napa 222
Amador 189
Madera 187
Lassen 178
Imperial 173
Santa Clara 167
Stanislaus 104
San Francisco 91
Contra Costa 85
Sonoma 75
Ventura 73
Del Norte 62
Shasta 56
Tulare 55
Tuolumne 48
Butte 45
Santa Barbara 44
San Mateo 31
Placer 29
Tehama 27
Yolo 26
Merced 25
Humboldt 25
Santa Cruz 24
El Dorado 20
Siskiyou 18
Mendocino 13
Lake 13
Nevada 8
Sutter 7
Yuba 6
Calaveras 4
Trinity 3
Glenn 3
Modoc 3
Inyo 3
Mariposa 2
San Benito 2
Mono 1
Colusa 1
Sierra 1



```python
# Set up the figure
fig = go.Figure()

# Add the death rate bar trace
fig.add_trace(go.Bar(
    y=mergedCounty.index,
    x=mergedCounty['DeathRate'],
    name='Death Rate',
    orientation='h',
    marker=dict(color='#2B2D42')
))

# Add the population bar trace
fig.add_trace(go.Bar(
    y=mergedCounty.index,
    x=mergedCounty['Population'],
    name='Population',
    orientation='h',
    marker=dict(color='#8D99AE')
))

# Update the layout
fig.update_layout(
    title='California County Death Rates and Populations',
    xaxis_title='Rate / Population',
    yaxis_title='County',
    barmode='group',
    bargap=0.1,
    height=800
)

#fig.write_html("/content/drive/MyDrive/BDA 600 - Project/horizontalbar_countyde
# Show the figure
fig.show()
```

```python
copyCustody = copy(mlcustody)
print(id(copyCustody))
print(id(mlcustody))
copyCustody.head(5)
```

```
# Organizing the Race Column

other_ethn = ['Other Asian',
              'Filipino',
              'Vietnamese',
              'Asian Indian',
              'Pacific Islander',
              'Korean',
              'Chinese',
              'Laotian',
              'Samoan',
              'Cambodian',
              'Japanese',
              'Guamanian',
              'Hawaiian', ]


copyCustody['race'] = copyCustody['race'].replace(other_ethn, "Asian/Pacific Isl

copyCustody = copyCustody[copyCustody['age'].str.isnumeric()]
copyCustody['age'] = copyCustody['age'].astype(str).astype(int)

copyCustody['race'].value_counts()
```

```
#Box Plot
fig = px.box(copyCustody, x='race', y='age', color='gender')
#fig.write_html("/content/drive/MyDrive/BDA 600 - Project/boxplot_ageracegender.
fig.show()
```

```
#Scatter Plot
fig = px.scatter(deathsEnvi, x='Unemployment', y='Poverty', color='DeathRate', s
#fig.write_html("/content/drive/MyDrive/BDA 600 - Project/scatterplot_countydeta
fig.show()
```

```
#Histogram
fig = px.histogram(mlcustody, x='age', nbins=20, marginal='box')
#fig.write_html("/content/drive/MyDrive/BDA 600 - Project/histogramboxplot_age.h
fig.show()
```

```
viz_data = merged_data

Asian_American = ['Other Asian',
              'Filipino',
              'Vietnamese',
              'Asian Indian',
              'Pacific Islander',
              'Korean',
              'Chinese',
              'Laotian',
              'Samoan',
              'Cambodian',
              'Japanese',
              'Guamanian',
      ]
Native_American = ["American Indian","Hawaiian"]

viz_data["race"]= viz_data["race"].replace(Native_American,"Native American")
viz_data["race"]= viz_data["race"].replace(Asian_American,"Asian American")
viz_data["race"]= viz_data["race"].replace("Black","African American")
viz_data["race"].value_counts()


print(viz_data[['Asian American (%)',
          'African American (%)',
          'Hispanic (%)',
          'Native American (%)',
          'Other/Multiple (%)',
          'White (%)', ]].mean())

# Define the population percentages by race, this information was taken from the
pop_percentages = {
    'Hispanic': 39.489592,
    'African American': 6.356513,
    'White': 39.905815,
    'Other': 3.409388,
    'Asian American': 10.298663,
    'Native American': 0.540274
}

viz_no_natural = viz_data[viz_data["manner_of_death"] != "Natural"]
viz_no_state = viz_data[viz_data["reporting_agency"] != "State"]
```

```
total_deaths_by_race = viz_data.groupby('race')['record_key_number'].count()
```

```
state_race_percentages = viz_data[['Asian American (%)',
          'African American (%)',
          'Hispanic (%)',
          'Native American (%)',
          'Other/Multiple (%)',
          'White (%)',
]].mean()
```

```
#Change name of Race (%) -> Race for proper concatination
state_race_percentages = state_race_percentages.rename({
    'African American (%)': 'African American',
    'Asian American (%)': 'Asian American',
    'Hispanic (%)': 'Hispanic',
    'Native American (%)': 'Native American',
    'Other/Multiple (%)': 'Other/Multiple',
    'White (%)': 'White'})

total_deaths_by_race = total_deaths_by_race.rename({'Other': 'Other/Multiple'})

race_percent = pd.concat((total_deaths_by_race, state_race_percentages), axis=1)
race_percent.rename(columns={'record_key_number': 'Total Deaths', 0: 'State Perc

race_percent
```

```
print(total_deaths_by_race)
print(" ")
print(state_race_percentages)
```

```
deaths_by_race = viz_data.groupby(['race'])[['record_key_number']].count()
deaths_by_race = deaths_by_race.reset_index()

deaths_by_race['pop_percentage'] = deaths_by_race['race'].map(pop_percentages)
deaths_by_race['normalized_deaths'] = deaths_by_race['record_key_number'] / (dea

deaths_by_race = deaths_by_race.sort_values("normalized_deaths", ascending = Fal

fig = px.bar(x=deaths_by_race["race"], y = deaths_by_race['normalized_deaths'])
#fig.write_html("/content/drive/MyDrive/BDA 600 - Project/bargraph_race_normaliz
fig.show()
```

Time Series Visualization

```
# Group the data by race and month
viz_data['month_of_death'] = viz_data['date_of_death'].dt.strftime('%Y-%m')
deaths_by_agency_and_month = viz_data.groupby(['custodial_responsibilty_at_time_
deaths_by_agency_and_month = deaths_by_agency_and_month.reset_index()

# Create an interactive line plot of the total number of deaths for each race ov
fig = go.Figure()
for race in deaths_by_agency_and_month['custodial_responsibilty_at_time_of_death
    df_race = deaths_by_agency_and_month[deaths_by_agency_and_month['custodial_r
    fig.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['record_key_

fig.update_layout(title='Monthly Deaths in Custody by Custodial Responsibly', xa
fig.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/death
fig.show()
```

```
# Group the data by race and month
viz_data['month_of_death'] = viz_data['date_of_death'].dt.strftime('%Y-%m')
deaths_by_agency_and_month = viz_data.groupby(['reporting_agency', 'month_of_dea
deaths_by_agency_and_month = deaths_by_agency_and_month.reset_index()

# Create an interactive line plot of the total number of deaths for each race ov
fig = go.Figure()
for race in deaths_by_agency_and_month['reporting_agency'].unique():
    df_race = deaths_by_agency_and_month[deaths_by_agency_and_month['reporting_a
    fig.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['record_key_

fig.update_layout(title='Monthly Deaths in Custody by Agency', xaxis_title='Year
fig.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/death
fig.show()



viz_no_natural['month_of_death'] = viz_no_natural['date_of_death'].dt.strftime('
deaths_by_agency_and_month = viz_no_natural.groupby(['reporting_agency', 'month_
deaths_by_agency_and_month = deaths_by_agency_and_month.reset_index()

# Create an interactive line plot of the total number of deaths for each race ov
fig2 = go.Figure()
for race in deaths_by_agency_and_month['reporting_agency'].unique():
    df_race = deaths_by_agency_and_month[deaths_by_agency_and_month['reporting_a
    fig2.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['record_key

fig2.update_layout(title='Monthly Deaths in Custody by Agency (No Natural Deaths
fig2.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/deat
fig2.show()
```

```
# Group the data by race and month
viz_data['month_of_death'] = viz_data['date_of_death'].dt.strftime('%Y-%m')
deaths_by_race_and_month = viz_data.groupby(['race', 'month_of_death'])[['recorc
deaths_by_race_and_month = deaths_by_race_and_month.reset_index()

# Normalize the number of deaths by population percentage for each race
deaths_by_race_and_month['pop_percentage'] = deaths_by_race_and_month['race'].ma
deaths_by_race_and_month['normalized_deaths'] = deaths_by_race_and_month['recorc

# Create an interactive line plot of the normalized number of deaths for each ra
fig = go.Figure()
for race in deaths_by_race_and_month['race'].unique():
    df_race = deaths_by_race_and_month[deaths_by_race_and_month['race'] == race]
    fig.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['normalized_

fig.update_layout(title='Monthly Deaths in Custody by Race', xaxis_title='Year',
#fig.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/deat

fig.show()

# Create an interactive line plot of the normalized number of deaths for each ra
fig2 = go.Figure()
for race in deaths_by_race_and_month['race'].unique():
    df_race = deaths_by_race_and_month[deaths_by_race_and_month['race'] == race]
    fig2.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['record_key

fig2.update_layout(title='Monthly Deaths in Custody by Race', xaxis_title='Year'
fig2.show()
```

```
# Group the data by race and month
viz_no_natural['month_of_death'] = viz_no_natural['date_of_death'].dt.strftime('
deaths_by_race_and_month = viz_no_natural.groupby(['race', 'month_of_death'])[['
deaths_by_race_and_month = deaths_by_race_and_month.reset_index()

# Normalize the number of deaths by population percentage for each race
deaths_by_race_and_month['pop_percentage'] = deaths_by_race_and_month['race'].ma
deaths_by_race_and_month['normalized_deaths'] = deaths_by_race_and_month['record

# Create an interactive line plot of the normalized number of deaths for each ra
fig = go.Figure()
for race in deaths_by_race_and_month['race'].unique():
    df_race = deaths_by_race_and_month[deaths_by_race_and_month['race'] == race]
    fig.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['normalized_

fig.update_layout(title='Monthly Deaths in Custody by Race (No Natrual Deaths)',
#fig.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/deat
fig.show()



# Create an interactive line plot of the normalized number of deaths for each ra
fig2 = go.Figure()
for race in deaths_by_race_and_month['race'].unique():
    df_race = deaths_by_race_and_month[deaths_by_race_and_month['race'] == race]
    fig2.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['record_key

fig2.update_layout(title='Monthly Deaths in Custody by Race (No Natrual Deaths)'
#fig2.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/dea
fig2.show()
```

```
# Group the data by race and month
viz_no_state['month_of_death'] = viz_no_state['date_of_death'].dt.strftime('%Y-%
deaths_by_race_and_month = viz_no_state.groupby(['race', 'month_of_death'])[['re
deaths_by_race_and_month = deaths_by_race_and_month.reset_index()

# Normalize the number of deaths by population percentage for each race
deaths_by_race_and_month['pop_percentage'] = deaths_by_race_and_month['race'].ma
deaths_by_race_and_month['normalized_deaths'] = deaths_by_race_and_month['record

# Create an interactive line plot of the normalized number of deaths for each ra
fig = go.Figure()
for race in deaths_by_race_and_month['race'].unique():
    df_race = deaths_by_race_and_month[deaths_by_race_and_month['race'] == race]
    fig.add_trace(go.Scatter(x=df_race['month_of_death'], y=df_race['normalized_

fig.update_layout(title='Monthly Deaths in Custody by Race (No State Arrests)',
#fig.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/inte
fig.show()
```

```
viz_data['month'] = viz_data['date_of_death'].dt.strftime('%Y-%m')
viz_data['week'] = viz_data['date_of_death'].dt.strftime('%Y-%U')
viz_data['year'] = viz_data['date_of_death'].dt.strftime('%Y')
```

```
# group data by week, year, and month
weekly_data = viz_data.groupby(['week', 'race'])['record_key_number'].count().re
yearly_data = viz_data.groupby(['year', 'race'])['record_key_number'].count().re
monthly_data = viz_data.groupby(['month', 'race'])['record_key_number'].count().

weekly_data['pop_percentage'] = weekly_data['race'].map(pop_percentages)
weekly_data['normalized_deaths'] = weekly_data['record_key_number'] / (weekly_da

yearly_data['pop_percentage'] = yearly_data['race'].map(pop_percentages)
yearly_data['normalized_deaths'] = yearly_data['record_key_number'] / (yearly_da

monthly_data['pop_percentage'] = monthly_data['race'].map(pop_percentages)
monthly_data['normalized_deaths'] = monthly_data['record_key_number'] / (monthly

# create an empty figure with dropdown menu to switch between different time fra
fig = go.Figure()
fig.update_layout(
    xaxis=dict(title='Years'),
    yaxis=dict(title='Number of Deaths (Normalized)')
)
```

```

# add traces for each race and time frame
for race in viz_data['race'].unique():
    weekly_trace = go.Scatter(
        x=weekly_data[weekly_data['race'] == race]['week'],
        y=weekly_data[weekly_data['race'] == race]['normalized_deaths'],
        mode='lines',
        name=race
    )
    fig.add_trace(weekly_trace)

    monthly_trace = go.Scatter(
        x=monthly_data[monthly_data['race'] == race]['month'],
        y=monthly_data[monthly_data['race'] == race]['normalized_deaths'],
        mode='lines',
        name=race
    )
    fig.add_trace(monthly_trace)

    yearly_trace = go.Scatter(
        x=yearly_data[yearly_data['race'] == race]['year'],
        y=yearly_data[yearly_data['race'] == race]['normalized_deaths'],
        mode='lines',
        name=race
    )
    fig.add_trace(yearly_trace)

# create the dropdown menu
updatemenus = [
    dict(
        buttons=list([
            dict(
                args=[{'visible': [True, False, False]}],#*len(viz_data['race'].
                label='Weekly',
                method='update'
            ),
            dict(
                args=[{'visible': [False, True, False]}],#*len(viz_data['race'].
                label='Monthly',
                method='update'
            ),
            dict(
                args=[{'visible': [False, False, True]}],#*len(viz_data['race'].
                label='Yearly',
                method='update'
            )
        ]),
        direction='down',
        showactive=True,
        xanchor='left',
        yanchor='top',
        active = 0
    )
]

fig.update_layout(
    title_text="Deaths in Custody by Race",
    updatemenus=updatemenus)

#fig.write_html("/content/drive/MyDrive/Grad School/BDA 600/BDA 600 Project/inte
fig.show()
```

```
# Group the data by day and count the number of deaths for each day
daily_deaths = viz_data.groupby('date_of_death')['record_key_number'].count()

# Calculate the rolling average of deaths using the Pandas rolling() function
rolling_avg = daily_deaths.rolling(window=20, min_periods=1).mean()

# Create a line graph of the rolling average using Plotly
fig = go.Figure()
fig.add_trace(go.Scatter(x=rolling_avg.index, y=rolling_avg.values, mode='lines'
fig.update_layout(title='Total Deaths: 20-Day Rolling Average', xaxis_title='Dat
                  title_x=0.5, font=dict(size=18), width=800, height=300,
                  hovermode='x', hoverlabel=dict(font_size=14),
                  legend=dict(x=0.01, y=0.98, borderwidth=1, font=dict(size=16))
fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey')
fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey')

# Set the tick label size
fig.update_layout(xaxis=dict(tickfont=dict(size=14)), yaxis=dict(tickfont=dict(s

fig.show()
```

```

viz_data2 = viz_data[viz_data["manner_of_death"]!="Natural"]

# Group the data by day and count the number of deaths for each day
daily_deaths = viz_data2.groupby('date_of_death')['record_key_number'].count()

# Calculate the rolling average of deaths using the Pandas rolling() function
rolling_avg = daily_deaths.rolling(window=20, min_periods=1).mean()

# Create a line graph of the rolling average using Plotly
fig = go.Figure()
fig.add_trace(go.Scatter(x=rolling_avg.index, y=rolling_avg.values, mode='lines'
fig.update_layout(title='Total Deaths: 20-Day Rolling Average (No Natural Deaths
                  title_x=0.5, font=dict(size=18), width=800, height=300,
                  hovermode='x', hoverlabel=dict(font_size=14),
                  legend=dict(x=0.01, y=0.98, borderwidth=1, font=dict(size=16))
fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey')
fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey')

# Set the tick label size
fig.update_layout(xaxis=dict(tickfont=dict(size=14)), yaxis=dict(tickfont=dict(s

fig.show()


```

Creating a New Export File of this dataset for Visualization purposes

A brief excerpt of code the develop a new CSV file of the manipulated data. This created a death rate per Race recorded for all of California.

```
#Exporting newly manipulated data for usage on other visualization tools
race_percent.to_csv('/content/drive/MyDrive/BDA 600 - Project/Race_State_DeathPe
```

```
#Exporting newly manipulated data for usage on other visualization tools
deaths_by_race.to_csv('/content/drive/MyDrive/BDA 600 - Project/Deaths_By_Race.c
```

Current Events Involving the Police [Near September 2017]

- August 12 – The Unite the Right rally, a gathering of alt-right, white nationalist, neo-Nazi, and neo-Confederate groups protesting the removal of the Robert Edward Lee Sculpture and other Confederate monuments and memorials from public spaces, is held in Charlottesville, Virginia. Violent clashes break out between attendees and counter-protesters; 32-year-old Heather Heyer is killed and many others are injured when a car ploughs into a group of people; and two Virginia State Police troopers are killed when their surveillance helicopter crashes, prompting Governor Terry McAuliffe to declare a state of emergency.
    - https://www.newsweek.com/white-nationalists-kkk-demo-virginia-647748
- August 28 – President Trump signs an executive order allowing police to acquire and use military-style equipment.
    - https://www.bbc.com/news/world-us-canada-41078158
- September 22 – During a political rally in Alabama, President Trump criticizes NFL football players kneeling during the national anthem in protest of police brutality against African-Americans, saying that team owners should "fire" them for doing it. The comments spark widespread condemnation and increases in protests from players during the national anthem.
    - https://www.cbsnews.com/news/trump-to-nfl-owners-fire-players-who-kneel-during-national-anthem/

```

```