CIS 4130

Jonathan Zhao
zhaojonathan456@gmail.com

Milestone 1: Proposal

Description

I plan to predict crime rates based on the relationship between median income and median age. Many factors influence crime, such as highest education level and health behaviors like binge drinking. I believe there is a strong relationship between the median income and median age for most people. How much money you earn is dependent on how much experience you have, and experience comes with time. In other words, there is an association with how much money you earn and how old you are. I collected data from the Data Commons Python API. I retrieved data about all 50 states in the United States and information for income, age, population, and crime for every year from 2011 to 2019. Some examples include median income of people who live in Wyoming in 2013 and the count of total crimes in Alabama in 2018. Each row represents a different year for one state, meaning there are 9 rows of data for each state.

Note: After exploring the data in Milestone 3, I realized that I did not have enough relevant information for a model. Specifically, I had data about crimes for different parts of the world, but most did not have a timestamp, like the year of when those numbers of crimes happened. Without several time stamps for each location, it would be difficult to yield any meaningful results from the model, since it wouldn't be comparable to a different time in the same location.

With the semester coming to an end, Professor Holowczak and I ultimately decided to stick to collecting the data that I needed, without worrying about gathering more than 10 GB of data, which was intended for this project as it gives a reason to use an engine for large-scale data processing like Spark. Furthermore, with a large amount of data, using a cloud computing service like AWS makes a lot more sense as it can do things like reduce costs, since you only pay for the services you use and you don't pay for the hardware that is used to process and deploy applications from the data. This led to the decision to use the current dataset from the Data Commons Python API. Previously, I was using data from the Data Commons CSV files, Data Commons Data Download Tool files, and a Kaggle dataset.

Milestone 2: Data Acquisition

- Amazon EC2 instance was created in the Management Console
- Connected to the Amazon EC2 instance and configured the AWS CLI with [access key]
  - Note: Do not forget to stop your instance if you are not using it
- Amazon S3 bucket was created using the AWS CLI

```
aws s3api create-bucket --bucket big-data-project-1 --region us-east-2
--create-bucket-configuration LocationConstraint=us-east-2
```

  where big-data-project-1 is the name of the bucket
- Objects (data) stored into the Amazon S3 bucket either in the Amazon EC2 instance using the AWS CLI or manually uploading the file to the Amazon S3 bucket, depending on where the data was coming from

Code for Data Commons Python API

Appendix A

- The resulting dataset needed to be transformed so that the columns could be used for a model, where every record is a yearly observation for a given state.
- A new column for state abbreviations was added to create visualizations of choropleth maps of U.S. states (Milestone 5)

	State	pop_2010	pop_2011	pop_2012	pop_2013	pop_2014	pop_2015	pop_2016	pop_2017	pop_2018	pop_2019	inc_2011	inc_2012
0	Alabama	4712651.0	4747424.0	4777326.0	4799277.0	4817678.0	4830620.0	4841164.0	4850771	4864680	4876250	22217	22318
1	Alaska	691189.0	700703.0	711139.0	720316.0	728300.0	733375.0	736855.0	738565	738516	737068	30604	31005
2	Arizona	6246816.0	6337373.0	6410979.0	6479703.0	6561516.0	6641928.0	6728577.0	6809946	6946685	7050299	26611	26388
3	Arkansas	2872684.0	2895928.0	2916372.0	2933369.0	2947036.0	2958208.0	2968472.0	2977944	2990671	2999370	21356	21604
4	California	36637290.0	36969200.0	37325068.0	37659181.0	38066920.0	38421464.0	38654206.0	38982847	39148760	39283497	27355	27129
5	Colorado	4887061.0	4966061.0	5042853.0	5119329.0	5197580.0	5278906.0	5359295.0	5436519	5531141	5610349	29921	30084
6	Connecticut	3545837.0	3558172.0	3572213.0	3583561.0	3592053.0	3593222.0	3588570.0	3594478	3581504	3575074	32910	32842
7	Delaware	881278.0	890856.0	900131.0	908446.0	917060.0	926454.0	934695.0	943732	949495	957248	29975	29752
8	Florida	18511620.0	18688787.0	18885152.0	19091156.0	19361792.0	19645772.0	19934451.0	20278447	20598139	20901636	25014	24683
9	Georgia	9468815.0	9600612.0	9714569.0	9810417.0	9907756.0	10006693.0	10099320.0	10201635	10297484	10403847	25828	25705
10	Hawaii	1333591.0	1346554.0	1362730.0	1376298.0	1392704.0	1406299.0	1413673.0	1421658	1422029	1422094	30378	30374

Sample of dataset from the API

	Year	State	Population	Income	Age	Crime	State_Abbreviation
0	2011	Alabama	4747424.0	22217	37.7	193364	AL
1	2012	Alabama	4777326.0	22318	37.8	190571	AL
2	2013	Alabama	4799277.0	22394	38.1	182819	AL
3	2014	Alabama	4817678.0	22626	38.2	174821	AL
4	2015	Alabama	4830620.0	22890	38.4	167698	AL
5	2016	Alabama	4841164.0	23527	38.6	169248	AL
6	2017	Alabama	4850771.0	24476	38.7	169711	AL
7	2018	Alabama	4864680.0	25375	38.9	163099	AL
8	2019	Alabama	4876250.0	26231	39.0	156179	AL
9	2011	Alaska	700703.0	30604	33.8	23411	AK
10	2012	Alaska	711139.0	31005	33.8	24449	AK

Sample of transformed dataset

<u>Amazon S3 Bucket with All Data Sources</u>

Check using this command: aws s3 ls s3://big-data-project-1

```
[ec2-user@ip-172-31-7-143 ~]$ aws s3 ls s3://big-data-project-1
2022-11-11 22:05:49 1854951641 datacommons_api.csv
```

Milestone 3: Exploratory Data Analysis

The dataset in the Amazon S3 bucket is loaded for exploratory data analysis using Python. The number of rows and columns, column names, data types of each column, number of missing values in each column, and descriptive statistics like mean and max for all the columns were found.

Notes:

- The code may paste in a way that makes it unreadable so double check it
- Place the code in a nano file: $ nano stats.py
- Run the nano file: $ python3 ./stats.py

```
# Loop through entire Amazon S3 bucket and get descriptive statistics for every CSV object (also works for every CSV object in folders)
import boto3
import pandas as pd
bucket_name="big-data-project-1"   # Put your bucket name here
s3_client = boto3.client('s3', use_ssl=False)
s3_resource = boto3.resource('s3')

def get_statistics(filename):
  # Do something here to get the statistics about the current file
  df = pd.read_csv(file_path, low_memory=False)
  print("Filename:", filename)
  print("\nNumber Rows and Columns:", df.shape)
  print("\nColumn Names:", df.columns)
  print("\nData Types:", df.dtypes)
  print("\nNumber Missing Values:", df.isnull().sum())
  print("\nMissing Values:", df[df.isnull().any(axis=1)])
  # Retrieve Descriptive Statistics and save as csv in the S3 bucket
  stats_filename = filename + "_stats.csv"
  df.describe(include='all').to_csv(stats_filename)
  df2 = pd.read_csv(stats_filename)
  print("\nDescriptive Statistics:", df2)

# Loop through objects in bucket
for object in s3_client.list_objects(Bucket=bucket_name)['Contents']:
  filename = object['Key']
```

```
if ".csv" in filename:
    print('Working on file name:', filename)
    # Create the full path to the file in the bucket
    file_path = "s3://" + bucket_name + "/" + filename
    # Call your function to analyze filename
    get_statistics(file_path)
```

```
Number Rows and Columns: (450, 8)
>>> print("\nColumn Names:", df.columns)

Column Names: Index(['Unnamed: 0', 'Year', 'State', 'Population', 'Income', 'Age', 'Crime',
       'State_Abbreviation'],
      dtype='object')
>>> print("\nData Types of Columns:", df.dtypes)

Data Types of Columns: Unnamed: 0              int64
Year                  float64
State                  object
Population            float64
Income                float64
Age                   float64
Crime                 float64
State_Abbreviation     object
dtype: object
>>> print("\nNumber Missing Values:", df.isnull().sum())

Number Missing Values: Unnamed: 0            0
Year                  0
State                 0
Population            0
Income                0
Age                   0
Crime                 0
State_Abbreviation    0
dtype: int64
>>> df_stats = df.describe(include='all')
>>> print("\nDescriptive Statistics:", df_stats)

Descriptive Statistics:          Unnamed: 0          Year     State    Population         Income          Age          Crime State_Abbreviation
count   450.000000   450.000000       450  4.500000e+02     450.000000  450.000000  4.500000e+02                450
unique         NaN          NaN        50           NaN            NaN         NaN           NaN                 50
top            NaN          NaN   Alabama           NaN            NaN         NaN           NaN                 AL
freq           NaN          NaN         9           NaN            NaN         NaN           NaN                  9
mean    224.500000  2015.000000       NaN  6.309457e+06   28031.611111   37.948000  1.853838e+05                NaN
std     130.048068     2.584863       NaN  7.004724e+06    3736.347478    2.337466  2.150661e+05                NaN
min       0.000000  2011.000000       NaN  5.546970e+05   20465.000000   29.100000  9.113000e+03                NaN
25%     112.250000  2013.000000       NaN  1.838038e+06   25320.500000   36.600000  4.655025e+04                NaN
50%     224.500000  2015.000000       NaN  4.466824e+06   27385.000000   38.000000  1.316485e+05                NaN
75%     336.750000  2017.000000       NaN  7.034090e+06   30742.500000   39.200000  2.259668e+05                NaN
max     449.000000  2019.000000       NaN  3.928350e+07   40341.000000   44.700000  1.210409e+06                NaN
```

Summary of Exploratory Data Analysis

In the data, each row is a specific year and gives information for a state in the United States for the total population, median income of a person, median age of a person, and total count of crimes. The data spans a total of 9 years, starting from 2011 to 2019. Furthermore, there is data for all 50 states. There are no missing values. From this analysis, I concluded that the dataset should be sufficient in predicting crime rates based on the relationship between median income and median age.

Milestone 4: Coding and Modeling

- Amazon EMR cluster was created in the Management Console
  - Use the latest Release (emr-6.9.0 as of writing this)
  - Select Spark as the Application
  - Choose the same EC2 key pair as the EC2 instance
- Configure the EMR cluster to allow SSH connections
  - After the EMR cluster has started, select it and click the Security Group for Master
  - Add an Inbound Rule for Port 22, SSH connection, and Anywhere IPv4
- Note: Terminate the EMR cluster once you are done using it and create a new cluster every time you use Amazon EMR

Logistic Regression

A logistic regression model predicts the probability (between 0 and 1) of an event. This is ideal for this dataset because in order to predict crime, there will need to be a measure to say whether or not it is good or bad. Assigning a 1 to represent crime that is higher than the national average and 0 for crime lower than the national average is perfect in this case. The features are state, population, income, and age. The label to predict is whether the total count of crime per 100,000 people in each state within a given year is over the national average of crime per 100,000 people or below for that given year.

Code for Logistic Regression Model

Appendix B

Main Steps of Code

1. PySpark reads the CSV dataset in the Amazon S3 bucket and creates a Spark DataFrame from it
2. Columns are created to measure crime: crime per 100,000 people for each year and state, national average number of crimes per 100,000 people for every year, and a column to create a label for the logistic regression model to work, whether the crime per 100,000 people for that state in that year is greater than the average national crime for that year
3. Feature Engineering was done for the state column: it was encoded and put in a VectorAssembler along with the Age, Population, and Income

4. A pipeline was created to standardize the data by applying the same transformations to the data at each step
5. The dataset was split where 70% became the training set and 30% became the test set for the logistic regression model
6. The Area Under the Curve (AUC) was used to evaluate the models. Steps to ensure the best model was picked included using a 3-Fold Validation and exploring the Hyperparameters (Grid Search) to see which model had the highest AUC
7. The best model was tested on the testing set

<u>Challenges when Cleaning and Processing the Data</u>
An issue that occurred while trying to prepare the date for modeling can be seen when trying to encode the columns Age, Population, and Income using MinMaxScaler. The purpose of using this is to scale the values to a 0.0 to 1.0 range because the values for these columns are based on different scales. For example, the Age column has values ranging from 29 to 44, while the Income column has values ranging from 20465 to 40341. Consequently, an error in one column may have a magnitude of 1%, while an error in another column has a different magnitude. After scaling these columns with the MinMaxScaler, placing them in the VectorAssembler makes them too large so PySpark runs out of memory.

An example why PySpark runs out of memory can be seen with an example like if you have a population of 4799277, then the vector will have at least 4799277 elements. Combine that with the Income and each vector will have millions of elements.

So instead the Age, Population, and Income columns were treated as a double data type. They were not encoded and were placed into the VectorAssembler directly as features.

Milestone 5: Visualizing Results

Visualizations of the data and prediction results were created with Spark tools and Python libraries (Matplotlib and Plotly Express).

Note: Spark has very few tools for data visualization, so a Spark DataFrame has to be converted to a Pandas DataFrame using .toPandas function in order to use plotting tools like Matplotlib and Seaborn. Additionally, visualization libraries typically assume a graphical user interface, such as Jupyter Notebook and Visual Studio Code, is being used. However, in a script there is no GUI so in order to view the plot, it needs to be saved somewhere like an Amazon S3 bucket.

Code for Visualizations

Appendix C

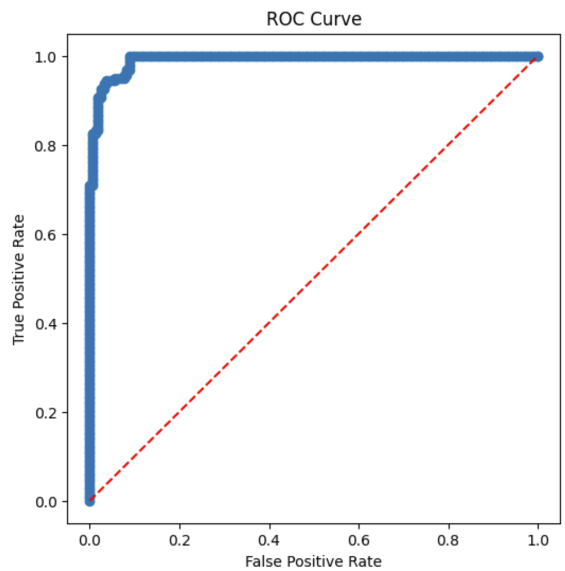
Confusion Matrix

highercrime	0.0	1.0
1.0	2	66
0.0	59	4

For the test set, the model predicted that the crime per 100,000 people:

- would be more than the national average when it was actually more (**True Positive**) 66 times.
- would be less than the national average when it was actually less (**True Negative**) 59 times.
- would be more than the national average when it was actually less (**False Positive**) 4 times.
- would be less than the national average when it was actually more (**False Negative**) 2 times.

ROC Curve



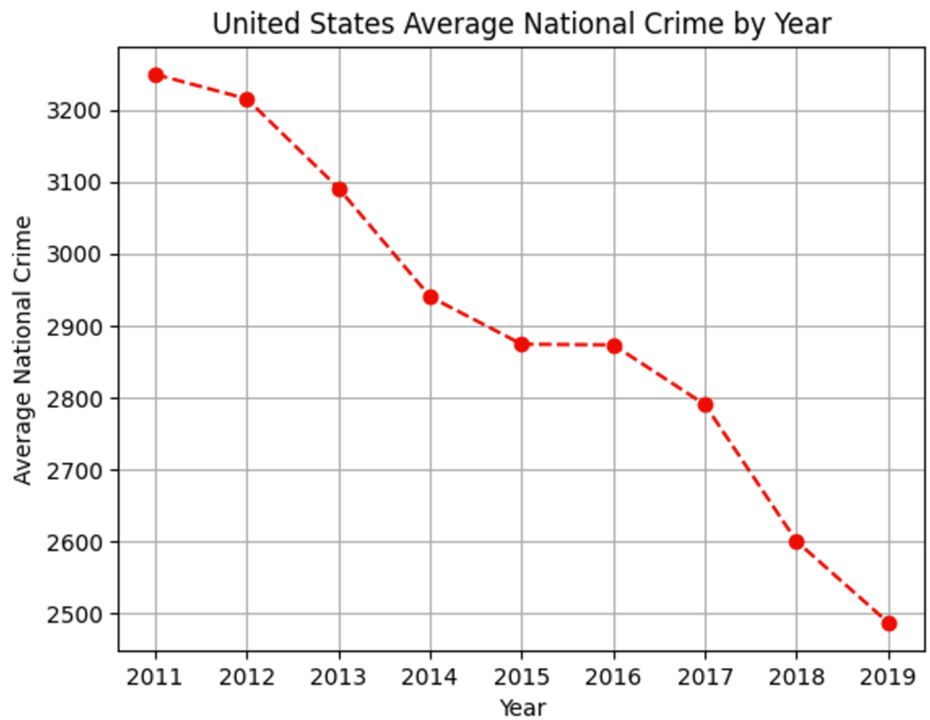
A plot for the **True Positive Rate** (observations correctly predicted) vs. **False Positive Rate** (observations incorrectly predicted) of the Logistic Regression model. Since the ROC curve is close to the top left corner of the graph, it means that the model has a good performance in predicting observations correctly.

Coefficients

```
0 stateVector_Alabama 1.0698836006522143
1 stateVector_Alaska 1.1028709961545915
2 stateVector_Arizona 1.0995410295834467
3 stateVector_Arkansas 1.064663061978985
4 stateVector_California 0.3039698933971121
5 stateVector_Colorado 0.13827453285489472
6 stateVector_Connecticut -0.8657942345543708
7 stateVector_Delaware 1.37294339927999
8 stateVector_Florida 1.19003475994689
9 stateVector_Georgia 0.98918950731532
10 stateVector_Hawaii 1.3605177136380482
11 stateVector_Idaho -1.4395441699363134
12 stateVector_Illinois -1.2596372495785366
13 stateVector_Indiana 0.28805200114149665
14 stateVector_Iowa -1.1445571600789528
15 stateVector_Kansas 1.0912935751184314
16 stateVector_Kentucky -1.2933838360904923
17 stateVector_Louisiana 1.0061399772894628
18 stateVector_Maine -0.9221074844301343
19 stateVector_Maryland -0.18736480834868077
20 stateVector_Massachusetts -0.9631002762568749
21 stateVector_Michigan -1.1938136460064874
22 stateVector_Minnesota -1.0924897753182838
23 stateVector_Mississippi 0.22208716907797102
24 stateVector_Missouri 1.1510417633919459
25 stateVector_Montana 0.44183194722587643
26 stateVector_Nebraska -1.2666023135070563
27 stateVector_Nevada 1.1638597484912003
28 stateVector_New Hampshire -0.8654832409157189
29 stateVector_New Jersey -0.9671914307087196
30 stateVector_New Mexico 1.047626915471883
31 stateVector_New York -1.2106496995201048
32 stateVector_North Carolina 1.0884723407701475
33 stateVector_North Dakota -1.2143333436771446
34 stateVector_Ohio 0.26584048932747667
35 stateVector_Oklahoma 1.0470218987079787
36 stateVector_Oregon 1.1657041820878313
37 stateVector_Pennsylvania -1.1247654053668183
38 stateVector_Rhode Island -1.0492270960399315
39 stateVector_South Carolina 1.1160541564108282
40 stateVector_South Dakota -1.2339726558333401
41 stateVector_Tennessee 1.106746634851878
42 stateVector_Texas 0.8357088767545624
43 stateVector_Utah 0.7843751657011515
44 stateVector_Vermont -0.9302520274328228
45 stateVector_Virginia -1.1158629379697396
46 stateVector_Washington 1.2711501647338803
47 stateVector_West Virginia -1.0558444207126276
48 stateVector_Wisconsin -1.1062995709686678
49 stateVector_Wyoming -1.1690030093646924
50 Population 1.3946663167282703e-08
51 Income -5.927376183855191e-05
52 Age -0.11510112353870883
```

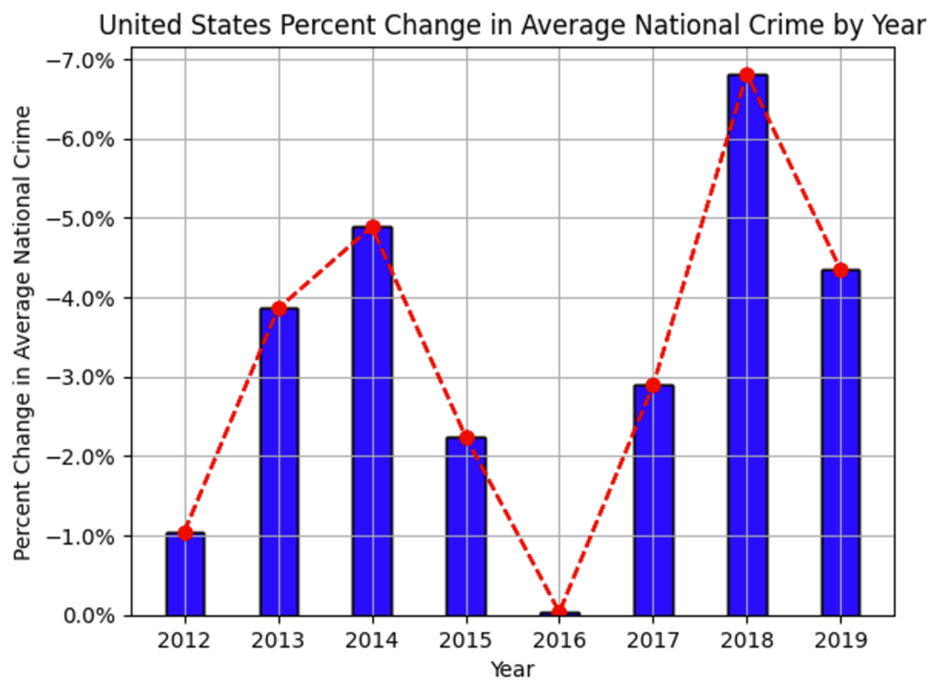
For each feature, it shows the effects on the model to identify what is important. For this Logistic Regression model, good **predictors** for predicting whether the average crime rate per 100,000 people for each state is greater than the national average from 2011-2019, include the population of people living in the state and living in certain states like Tennessee.

Average National Crime Over Time



A line graph showing the average national crime per 100,000 people over time in the United States from 2011 to 2019. With each increasing year, **crime decreased** (2015 had 2874 crimes and 2016 had 2873 crimes). Compared to the highest crime rate in 2011 with 3249 crimes, the lowest crime rate in 2019 was 2487.

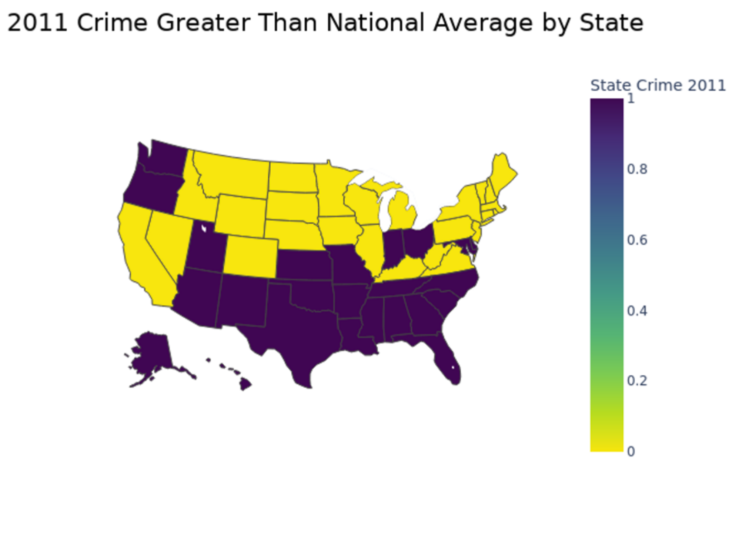
Percent Change in Average National Crime

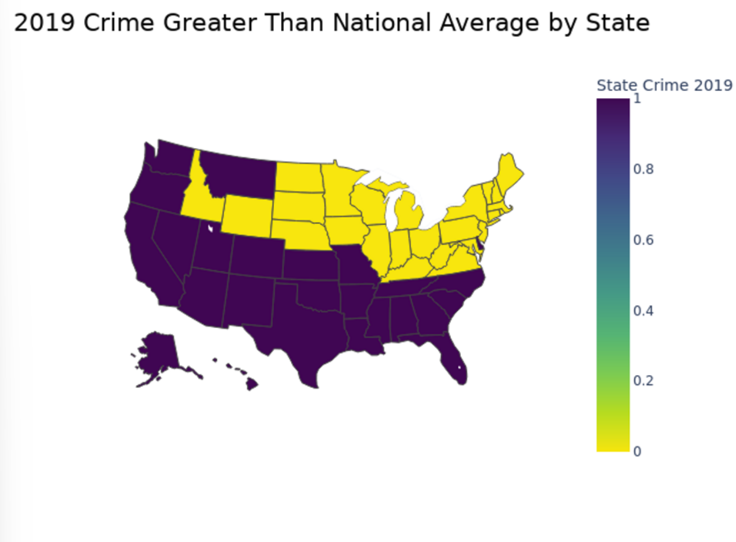


Note: The percent change value for the year 2012 is obtained from getting the percent difference between the crime rate in 2011 and 2012.

A bar graph with a line showing **how much the average national crime** per 100,000 people in the United States changes (**decreases**) as a percent from 2011 to 2019. Based on the previous graph, it is known that the crime rate with each increase in year declined, so all the values for the percent change are negative. An example of interpreting the graph is in 2013, the crime rate had an almost 4% decrease from 2012.

Year Over Year National Average Crime

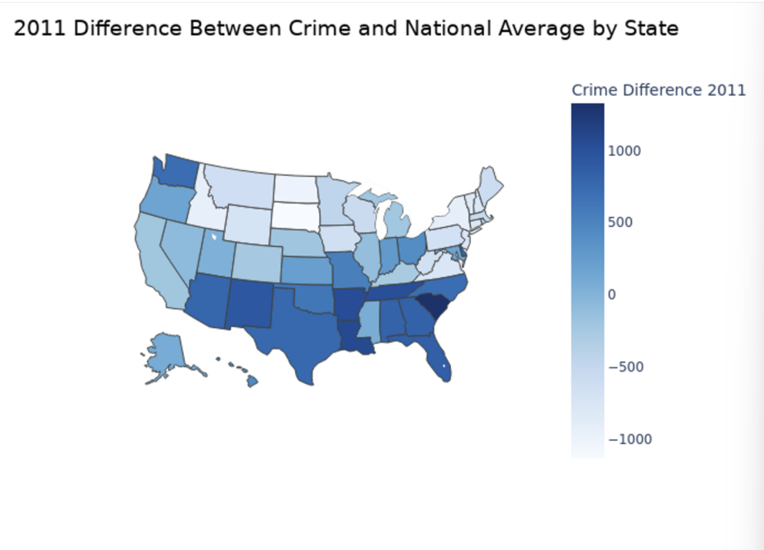


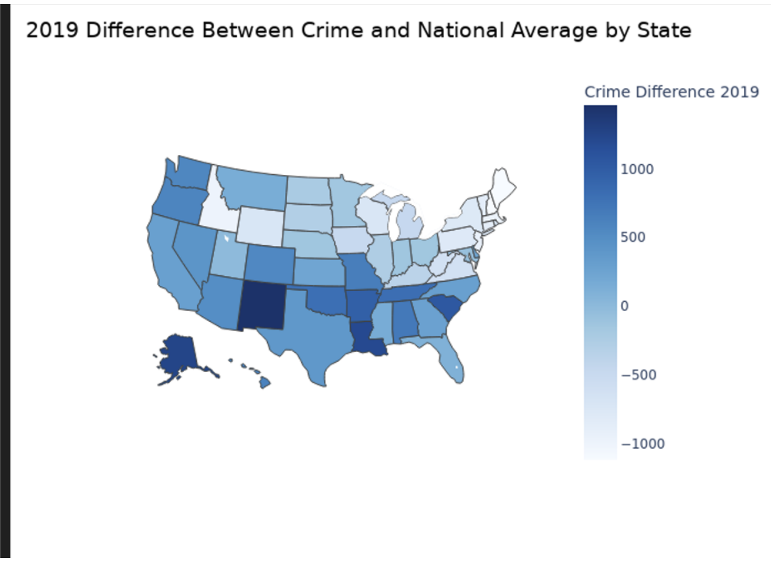


Note: Purple indicates crime is above the national average, whereas yellow indicates crime is under the national average.

Plots comparing whether the crime rate per 100,000 people for each state is greater than the national average in 2011 vs in 2019. In 2011, states like California and Montana had crime under the national average, whereas those same states in 2019 are above the national average. **Depending on the year**, certain states are above the average nation crime, while others are below.

Year Over Year Change in Crime





Note: A darker shade of blue (positive numbers) indicates crime in that state is higher than the national average, with the shade of blue showing how much higher it is. Whereas a lighter shade of the blue (negative numbers) indicates crime in that state is lower than the national average, with the shade of blue showing how much lower it is.

Plots that take the difference between crime per 100,000 people and national average for 2011 and 2019 respectively. A state with a number near 1000 like Louisiana and Arkansas in 2011 have higher crime rates, compared to a state with a number near -1000 like South Dakota in 2011 have lower crime rates. Comparing the plots for 2011 and 2019, it can be seen that in those two years, states like Ohio and Florida have had a decline in crime, whereas states like Alaska and New York have had a rise in crime, so there is a mix between **crime rising and declining depending on the state**.

Milestone 6: Summary and Conclusions

Code of Complete Pipeline
Appendix D

Model Results

- The best model had the best performance. The **Area Under the ROC Curve** (AUC) was used to evaluate this, where values range from 0 - 1. Scoring a 1 means the model is perfect, whereas scoring a 0 means all the predictions were wrong.
- To validate that the model was not a result from the random split, a **3-Fold Validation** was used on the training data. This runs the model 3 times, where the data is split into 3 parts and ⅔ of data is built for the model while ⅓ is held off for each split. The average AUC over these models was 0.9619. The AUC was then used for the testing data for a score of 0.9872.
- To optimize the model, a range of **Hyperparameters**, parameters that are fixed and can affect how well a model trains, were explored by carrying out multiple splits and then seeing which parameters lead to the best model performance, also known as **Grid Search**.
    - As a result, **regparams** were used, which are hyperparameters that try to prevent a model from **overfitting**, where the model performs well on the training data, but not on new, unseen data . Six of them were used to specify the range of regparam values to use when searching for the best model hyperparameters. In this case, they were 0.0, 0.2, 0.4, 0.6, 0.8, 1.0.
    - Additionally, 2 **elasticNetParams** of 0 (Ridge Regression) and 1 (Lasso regression) discourage the model from learning complex and overfitted models, resulting in 12 different models to be tested.
    - Those 12 models with the 3 number of folds, resulted in 36 total models. Each model was tested on the performance (AUC) for each combination and the combination with the best performance was selected.
- The **best model** had an AUC of 0.9921, which is almost perfect.
- This best model was tested on the testing set, resulting in an AUC of 0.9820.

Next Steps

In the future, I would like to investigate deeper into the counties, cities, and zip codes within these states to learn more about crime rates and what features help predict it. Since there would be more data to collect and process, it would allow using an engine for large-scale data processing like Spark and a cloud computing service like AWS to be used as intended. In addition, I would like to use different software to create data visualizations. A **business intelligence tool such as Tableau** can connect to multiple data sources like a database or CSV file at the same time and use them all to build visualizations by data blending. I could build the same visualizations and more with its simple drag and drop functionality. Afterwards, assembling all the visualizations into a dashboard and presenting these insights would be effective in telling the story to the viewer.

Milestone 7: GitHub Links

Project website

Project repository

Code Examples

1. Transform the columns of the dataset to use for a model, so that every record is a yearly observation for a given state: Year, State, Population, Income, Age by Professor Holowczak

```
transformed_df = pd.DataFrame()
years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019]
for index, row in final.iterrows():
   for year in years:
      syear = str(year)
      year, row['State'], row['pop_'+syear], row['inc_'+syear], row['age_'+syear], row['crime_'+syear]
      # Assemble a new record
      transformed_df = transformed_df.append({'Year': year, 'State': row['State'], 'Population': row['pop_'+syear],
'Income':row['inc_'+syear],'Age': row['age_'+syear], 'Crime':row['crime_'+syear] }, ignore_index=True)
```

2. Add state abbreviations to the dataset to be used in creating map visualizations of the United States by Professor Holowczak

```
us_state_to_abbrev = {
   "Alabama": "AL",    "Alaska": "AK",    "Arizona": "AZ",    "Arkansas": "AR",    "California": "CA",
   "Colorado": "CO",    "Connecticut": "CT",    "Delaware": "DE",    "Florida": "FL",    "Georgia": "GA",
   "Hawaii": "HI",    "Idaho": "ID",    "Illinois": "IL",    "Indiana": "IN",    "Iowa": "IA",    "Kansas": "KS",
   "Kentucky": "KY",    "Louisiana": "LA",    "Maine": "ME",    "Maryland": "MD",    "Massachusetts": "MA",
   "Michigan": "MI",    "Minnesota": "MN",    "Mississippi": "MS",    "Missouri": "MO",    "Montana": "MT",
   "Nebraska": "NE",    "Nevada": "NV",    "New Hampshire": "NH",    "New Jersey": "NJ",    "New Mexico": "NM",
   "New York": "NY",    "North Carolina": "NC",    "North Dakota": "ND",    "Ohio": "OH",    "Oklahoma": "OK",
   "Oregon": "OR",    "Pennsylvania": "PA",    "Rhode Island": "RI",    "South Carolina": "SC",    "South Dakota":
"SD",
   "Tennessee": "TN",    "Texas": "TX",    "Utah": "UT",    "Vermont": "VT",    "Virginia": "VA",    "Washington":
"WA",
   "West Virginia": "WV",    "Wisconsin": "WI",    "Wyoming": "WY",    "District of Columbia": "DC",    "American
Samoa": "AS",
   "Guam": "GU",    "Northern Mariana Islands": "MP",    "Puerto Rico": "PR",    "United States Minor Outlying
Islands": "UM",
   "U.S. Virgin Islands": "VI"
}

transformed_df['State_Abbreviation'] = transformed_df['State'].map(us_state_to_abbrev)
```

3. Saving a plot to Amazon S3 by Professor Holowczak

```
import io
import matplotlib.pyplot as plt
import s3fs # install with pip3 install s3fs
# Create a plot
plt.plot(some figure)
# Create a buffer to hold the figure
```

```
img_data = io.BytesIO()
# Write the figure to the buffer plt.savefig(img_data, format='png', bbox_inches='tight') img_data.seek(0)
# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://my-data-bucket/my_plot.png', 'wb') as f:
  f.write(img_data.getbuffer())
```

4. Confusion Matrix by Professor Holowczak

```
# Test the predictions
predictions = cv.transform(testData)
# Calculate AUC
auc = evaluator.evaluate(predictions)
print('AUC:', auc)
# Create the confusion matrix predictions.groupby('label').pivot('prediction').count().fillna(0).show()
cm = predictions.groupby('label').pivot('prediction').count().fillna(0).collect()
def calculate_recall_precision(cm):
  tn = cm[0][1] # True Negative
  fp = cm[0][2] # False Positive
  fn = cm[1][1] # False Negative
  tp = cm[1][2] # True Positive
  precision = tp / ( tp + fp )
  recall = tp / ( tp + fn )
  accuracy = ( tp + tn ) / ( tp + tn + fp + fn )
  f1_score = 2 * ( ( precision * recall ) / ( precision + recall ) )
  return accuracy, precision, recall, f1_score
print( calculate_recall_precision(cm) )
```

5. ROC Curve in Matplotlib by Professor Holowczak

```
# Look at the parameters for the best model that was evaluated from the grid
parammap = cv.bestModel.stages[3].extractParamMap()
for p, v in parammap.items():
  print(p, v)
# Grab the model from Stage 3 of the pipeline
mymodel = cv.bestModel.stages[3]
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5))
plt.plot(mymodel.summary.roc.select('FPR').collect(), mymodel.summary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
plt.savefig("roc1.png")
```

6. Coefficients of each variable by Professor Holowczak

```
# Extract the coefficients on each of the variables
coeff = mymodel.coefficients.toArray().tolist()
```

```
# Loop through the features to extract the original column names. Store in the var_index dictionary
var_index = dict()
for variable_type in ['numeric', 'binary']:
    for variable in predictions.schema["features"].metadata["ml_attr"]["attrs"][variable_type]:
        print("Found variable:", variable)
        idx = variable['idx']
        name = variable['name']
        var_index[idx] = name # Add the name to the dictionary
# Loop through all of the variables found and print out the associated coefficients
for i in range(len(var_index)):
    print(i, var_index[i], coeff[i])
```

7. Create a map of the United States in Python

Appendix A

Request Data From the Data Commons Python API and Load Into an Amazon S3 Bucket as a File

1. Connect to Amazon EC2 instance as an ec2-user and use the AWS CLI
2. Make sure the Python version is at least 3.7 (requirement at the time of writing this document): $ python 3 --version
3. Install Boto3 and other modules: $ pip3 install boto3 pandas fsspec s3fs
4. Use python3: $ python3
5. Use the script below (to replicate, change the bucket and file name when saving to csv)

Notes:

- The code may paste in a way that causes an error so double check it
- Place the code in a nano file: $ nano dc_api.py
- Run the nano file: $ python3 ./dc_api.py

```
# Data Commons Python API
# Run in AWS CLI

import datacommons_pandas as dc
import pandas as pd
#import boto3      # remove comment when running code in AWS CLI

# In the browser, we saw that the dcid for United States is country/USA
usa = 'country/USA'
# The Pandas API defines a number of convenience functions for building Pandas DataFrames with information in
the
# datacommons graph. We will be using get_places_in which requires three arguments:
   # dcids - A list or pandas.Series of dcids identifying administrative areas that we wish to get containing places for.
   # place_type - The type of the administrative area that we wish to query for.

states = dc.get_places_in([usa], 'State')[usa]



# population column

pop = dc.build_time_series_DataFrame(states, 'Count_Person')

# drop all rows with missing values
pop.dropna(inplace= True)

# rename year columns
pop.rename(columns={"2010":"pop_2010","2011":"pop_2011","2012":"pop_2012","2013":"pop_2013","2014":"p
op_2014",
            "2015":"pop_2015","2016":"pop_2016","2017":"pop_2017","2018":"pop_2018","2019":"pop_2019"},
      inplace=True)
```

```
# add States column
# To get the name of the state, we can use the get_property_values function:
def add_name_col(df):
   # Add a new column called name, where each value is the name for the place dcid in the index.
   df['name'] = df.index.map(dc.get_property_values(df.index, 'name'))

   # Keep just the first name, instead of a list of all names.
   df['name'] = df['name'].str[0]

add_name_col(pop)
pop.index.names = ['State_ID']
pop.rename(columns = {'name':'State'}, inplace=True)

#reorder State column
state_cols = ['State']
state_new_columns = state_cols + (pop.columns.drop(state_cols).tolist())
pop = pop[state_new_columns]



# median income person column

inc = dc.build_time_series_DataFrame(states, 'Median_Income_Person')

# drop all rows with missing values
inc.dropna(inplace= True)

# rename year columns
inc.rename(columns={"2011":"inc_2011","2012":"inc_2012","2013":"inc_2013","2014":"inc_2014",
               "2015":"inc_2015","2016":"inc_2016","2017":"inc_2017","2018":"inc_2018","2019":"inc_2019",
               "2020":"inc_2020"}, inplace=True)


# add States column
# To get the name of the state, we can use the get_property_values function:
def add_name_col(df):
   # Add a new column called name, where each value is the name for the place dcid in the index.
   df['name'] = df.index.map(dc.get_property_values(df.index, 'name'))

   # Keep just the first name, instead of a list of all names.
   df['name'] = df['name'].str[0]

add_name_col(inc)
inc.index.names = ['State_ID']
inc.rename(columns = {'name':'State'}, inplace=True)
```

```
#reorder State column
state_cols = ['State']
state_new_columns = state_cols + (inc.columns.drop(state_cols).tolist())
inc = inc[state_new_columns]



# median age person column

age = dc.build_time_series_DataFrame(states, 'Median_Age_Person')

# drop all rows with missing values
age.dropna(inplace= True)

# rename year columns
age.rename(columns={"2010":"age_2010","2011":"age_2011","2012":"age_2012","2013":"age_2013","2014":"age
_2014",
            "2015":"age_2015","2016":"age_2016","2017":"age_2017","2018":"age_2018","2019":"age_2019"},
     inplace=True)


# add States column
# To get the name of the state, we can use the get_property_values function:
def add_name_col(df):
  # Add a new column called name, where each value is the name for the place dcid in the index.
  df['name'] = df.index.map(dc.get_property_values(df.index, 'name'))

  # Keep just the first name, instead of a list of all names.
  df['name'] = df['name'].str[0]

add_name_col(age)
age.index.names = ['State_ID']
age.rename(columns = {'name':'State'}, inplace=True)

#reorder State column
state_cols = ['State']
state_new_columns = state_cols + (age.columns.drop(state_cols).tolist())
age = age[state_new_columns]



# crimes column

crime = dc.build_time_series_DataFrame(states, 'Count_CriminalActivities_CombinedCrime')

# drop all rows with missing values
```

```
crime.dropna(inplace= True)

# rename year columns
crime.rename(columns={"2008":"crime_2008","2009":"crime_2009","2010":"crime_2010","2011":"crime_2011","
2012":"crime_2012",

"2013":"crime_2013","2014":"crime_2014","2015":"crime_2015","2016":"crime_2016","2017":"crime_2017",
                "2018":"crime_2018","2019":"crime_2019"}, inplace=True)


# add States column
# To get the name of the state, we can use the get_property_values function:
def add_name_col(df):
   # Add a new column called name, where each value is the name for the place dcid in the index.
   df['name'] = df.index.map(dc.get_property_values(df.index, 'name'))

   # Keep just the first name, instead of a list of all names.
   df['name'] = df['name'].str[0]

add_name_col(crime)
crime.index.names = ['State_ID']
crime.rename(columns = {'name':'State'}, inplace=True)

#reorder State column
state_cols = ['State']
state_new_columns = state_cols + (crime.columns.drop(state_cols).tolist())
crime = crime[state_new_columns]



# merge all DataFrames together
df = pd.merge(pop,inc,on='State')
df2 = pd.merge(df,age,on='State')
final = pd.merge(df2,crime,on='State')

pd.set_option('display.max_columns', None)
final



# Instead of having every column be a field with the year (pop_2011, pop_2012), transform the columns to use for
a model,
# so that every record is a yearly observation for a given state: Year, State, Population, Income, Age


transformed_df = pd.DataFrame()
```

```
years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019]
for index, row in final.iterrows():
    for year in years:
       syear = str(year)
       year, row['State'], row['pop_'+syear], row['inc_'+syear], row['age_'+syear], row['crime_'+syear]
       # Assemble a new record
       transformed_df = transformed_df.append({'Year': year, 'State': row['State'], 'Population': row['pop_'+syear],
'Income':row['inc_'+syear],'Age': row['age_'+syear], 'Crime':row['crime_'+syear] }, ignore_index=True)



# Need state abbreviations to create visualizations of choropleth maps of U.S. states

us_state_to_abbrev = {
    "Alabama": "AL",     "Alaska": "AK",     "Arizona": "AZ",     "Arkansas": "AR",     "California": "CA",
    "Colorado": "CO",     "Connecticut": "CT",     "Delaware": "DE",     "Florida": "FL",     "Georgia": "GA",
    "Hawaii": "HI",     "Idaho": "ID",     "Illinois": "IL",     "Indiana": "IN",     "Iowa": "IA",     "Kansas": "KS",
    "Kentucky": "KY",     "Louisiana": "LA",     "Maine": "ME",     "Maryland": "MD",     "Massachusetts": "MA",
    "Michigan": "MI",     "Minnesota": "MN",     "Mississippi": "MS",     "Missouri": "MO",     "Montana": "MT",
    "Nebraska": "NE",     "Nevada": "NV",     "New Hampshire": "NH",     "New Jersey": "NJ",     "New Mexico": "NM",
    "New York": "NY",     "North Carolina": "NC",     "North Dakota": "ND",     "Ohio": "OH",     "Oklahoma": "OK",
    "Oregon": "OR",     "Pennsylvania": "PA",     "Rhode Island": "RI",     "South Carolina": "SC",     "South Dakota":
"SD",
    "Tennessee": "TN",     "Texas": "TX",     "Utah": "UT",     "Vermont": "VT",     "Virginia": "VA",     "Washington":
"WA",
    "West Virginia": "WV",     "Wisconsin": "WI",     "Wyoming": "WY",     "District of Columbia": "DC",     "American
Samoa": "AS",
    "Guam": "GU",     "Northern Mariana Islands": "MP",     "Puerto Rico": "PR",     "United States Minor Outlying
Islands": "UM",
    "U.S. Virgin Islands": "VI"
}


transformed_df['State_Abbreviation'] = transformed_df['State'].map(us_state_to_abbrev)


# Save final dataset as CSV to Amazon S3 bucket (remove comment when running code in AWS CLI)
#transformed_df.to_csv('s3://big-data-project-1/datacommons_api.csv')     #change "bucket-name" and
"file-name" to yours


# Dataset for model
transformed_df
```

Appendix B

Logistic Regression Model using PySpark on Amazon EMR

1. Create an Amazon EMR cluster
2. Once it is ready to use, click on the cluster name and identify the Master public DNS number
3. On the Amazon EC2 console, locate the instance with the same Public DNS (IPv4) number
4. Connect to that instance with User name hadoop
5. Use pyspark: $ pyspark
6. Use the script below (to replicate, change the bucket and file name)

Notes:

- The code may paste in a way that causes an error so double check it

```
sc.setLogLevel("ERROR")


# Import some functions we will need later on
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
import pyspark.ml.evaluation as evals
import pyspark.ml.tuning as tune
import numpy as np


# Set up the path to the final dataset in S3 bucket
bucket = 'big-data-project-1/'
filename = 'datacommons_api.csv'
file_path = 's3a://' + bucket + filename

# Create a Spark DataFrame from the file on S3
sdf = spark.read.csv(file_path, header=True, inferSchema=True)

# create column for crime per 100,000 people
# floor function rounds the calculation down
sdf = sdf.withColumn('Crime_Per_Hundred_Thousand', floor(sdf.Crime / sdf.Population * 100000))

# show some data
sdf_select = sdf.select('State','Year','Crime','Population','Crime_Per_Hundred_Thousand').show(10)

# calculate national average number of crimes per 100,000 people for each year
average_crime_sdf = sdf.groupBy('Year').agg(round(mean('Crime_Per_Hundred_Thousand')).alias('average_crime'))

# Show the average crime
average_crime_sdf.show(10)
```

```
# Join the average crime DataFrame back into the original DataFrame sdf
sdf = sdf.join(average_crime_sdf, "Year")

# Show some of the new, joined data
sdf.select('State','Year','Crime','Population','Crime_Per_Hundred_Thousand','average_crime').show(10)


# Create the label, =1 if Crime_per_Hundred_Thousand > average_crime, =0 otherwise
sdf = sdf.withColumn("highercrime", when(sdf.Crime_Per_Hundred_Thousand > sdf.average_crime,
1.0).otherwise(0.0))

# Show some of the data with the label
sdf_select =
sdf.select('State','Year','Crime','Population','Crime_per_Hundred_Thousand','average_crime','highercrime').show(1
0)



# Prepare data for modeling

# Feauture Engineering

# Data types of columns
sdf.printSchema()

# Trying to encode the features: age, population, income using MinMaxScaler to scale the values to a 0.0 to 1.0
range and then putting them in the vector assembler makes them too large and PySpark runs out of memory
# For example, if you have a population of 4799277, then the vector will have at least 4799277 elements. Combine
that with the Income and each vector will have millions of elements
# Instead will treat age, population, income as double and without encoding them, place into vector assembler
directly as features


# change Income to double
sdf = sdf.withColumn("Income", sdf.Income.cast("double"))


# the feature State is a string


# StringIndexer
indexer = StringIndexer(inputCols=['State'], outputCols=['stateIndex'])


# OneHotEncoder
```

```
encoder = OneHotEncoder(inputCols=['stateIndex'], outputCols=['stateVector'], dropLast=False)


# Vector Assembler
assembler = VectorAssembler(inputCols=['stateVector','Population','Income', 'Age'],
outputCol="features")


# Create pipeline
crime_pipe = Pipeline(stages=[indexer, encoder, assembler])

# Call .fit to transform the data
transformed_sdf = crime_pipe.fit(sdf).transform(sdf)

# Review the transformed features
transformed_sdf.select('State','stateVector','Year','Population','Income','Age','Crime_per_Hundred_Thousand','aver
age_crime','highercrime','features').show(20, truncate=False)

# Split data
train, test = transformed_sdf.randomSplit([.7, .3], seed=3456)

# LogisticRegression
lr = LogisticRegression(labelCol="highercrime")

# Fit the model
model = lr.fit(train)

# Show model coefficients and intercept
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)

# Test the model on the test data
test_results = model.transform(test)

# Test Results

# Show the test results
test_results.select('State','Year','Population','Income','Age','Crime_per_Hundred_Thousand','average_crime','rawPr
ediction','probability','prediction','highercrime').show(truncate=False)

# Show the confusion matrix
test_results.groupby('highercrime').pivot('prediction').count().show()


# Model Validation
```

```
# Create a BinaryClassificationEvaluator to evaluate how well the model works
evaluator = evals.BinaryClassificationEvaluator(labelCol="highercrime", metricName="areaUnderROC")

# Create the parameter grid (empty for now)
grid = tune.ParamGridBuilder().build()

# Create the CrossValidator
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3, seed=789)

# Use the CrossValidator to Fit the train data
cv = cv.fit(train)

# Show the average performance over the three folds
cv.avgMetrics


# Evaluate the test data using the cross-validator model
# Reminder: We used Area Under the Curve
evaluator.evaluate(cv.transform(test))


# Tuning
# Explore a range of Hyperparameters, carry out multiple splits and then see which parameters lead to the best model performance

# Create a grid to hold hyperparameters
# Logistic Regression threshold from 0 to 0.1 in .01 increments

grid = tune.ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid
grid = grid.build()
print('Number of models to be tested: ', len(grid))

# Create the CrossValidator using the new hyperparameter grid
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)

# Call cv.fit() to create models with all of the combinations of parameters in the grid
all_models = cv.fit(train)

print("Average Metrics for Each model: ", all_models.avgMetrics)
```

```
# Get the best model

# Gather the metrics and parameters of the model with the best average metrics
hyperparams = all_models.getEstimatorParamMaps()[np.argmax(all_models.avgMetrics)]

# Print out the list of hyperparameters for the best model
for i in range(len(hyperparams.items())):
   print([x for x in hyperparams.items()][i])

(Param(parent='LogisticRegression_daf2ca402a50', name='regParam', doc='regularization parameter (>= 0).'), 1.0)
(Param(parent='LogisticRegression_daf2ca402a50', name='elasticNetParam', doc='the ElasticNet mixing parameter, in
range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'), 0.0)

# Choose the best model
bestModel = all_models.bestModel
print("Area under ROC curve:", bestModel.summary.areaUnderROC)    # Area under ROC curve: 0.99205912414498


# Test the best model on the test set

# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test)

# Show the results
test_results.select('stateVector','Population','Income', 'Age', 'probability', 'prediction',
'highercrime').show(truncate=False)

# Evaluate the predictions. Area Under ROC curve
print(evaluator.evaluate(test_results))    # 0.9820261437908496
```

Appendix C

Visualizations for the data and model results using PySpark on Amazon EMR

1. Create an Amazon EMR cluster
2. Once it is ready to use, click on the cluster name and identify the Master public DNS number
3. On the Amazon EC2 console, locate the instance with the same Public DNS (IPv4) number
4. Connect to that instance with User name hadoop
5. Each time you create a new EMR cluster, have to install these packages again:
    a. $ pip3 install s3fs
    b. $ pip3 install matplotlib
    c. $ pip3 install plotly-express
    d. $ pip3 install kaleido
6. Use pyspark: $ pyspark
7. Use the script below (to replicate, change the bucket and file name)

Notes:

- The code may paste in a way that causes an error so double check it

```
sc.setLogLevel("ERROR")


# Import some functions we will need later on
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
import pyspark.ml.evaluation as evals
import pyspark.ml.tuning as tune
import pandas as pd
import numpy as np
import io
import s3fs
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
import plotly.express as px


# Set up the path to the final dataset in S3 bucket
bucket = 'big-data-project-1/'
filename = 'datacommons_api_abbreviations.csv'
file_path = 's3a://' + bucket + filename

# Create a Spark DataFrame from the file on S3
sdf = spark.read.csv(file_path, header=True, inferSchema=True)

# create column for crime per 100,000 people
```

```
# floor function rounds the calculation down
sdf = sdf.withColumn('Crime_Per_Hundred_Thousand', floor(sdf.Crime / sdf.Population * 100000))

# show some data
sdf_select = sdf.select('State','Year','Crime','Population','Crime_Per_Hundred_Thousand').show(10)

# calculate national average number of crimes per 100,000 people for each year
average_crime_sdf = sdf.groupBy('Year').agg(round(mean('Crime_Per_Hundred_Thousand')).alias('average_crime'))


# Show the average crime
average_crime_sdf.show(10)

# Join the average crime DataFrame back into the original DataFrame sdf
sdf = sdf.join(average_crime_sdf, "Year")

# Show some of the new, joined data
sdf.select('State','Year','Crime','Population','Crime_Per_Hundred_Thousand','average_crime').show(10)


# Create the label, =1 if Crime_per_Hundred_Thousand > average_crime, =0 otherwise
sdf = sdf.withColumn("highercrime", when(sdf.Crime_Per_Hundred_Thousand > sdf.average_crime,
1.0).otherwise(0.0))


# (for visualization) Create the label, =1 if Crime_per_Hundred_Thousand > average_crime, =0 otherwise
sdf = sdf.withColumn("State Crime", when(sdf.Crime_Per_Hundred_Thousand > sdf.average_crime,
1.0).otherwise(0.0))


# Show some of the data with the label
sdf_select =
sdf.select('State','Year','Crime','Population','Crime_per_Hundred_Thousand','average_crime','highercrime').show(1
0)



# Prepare data for modeling

# Feauture Engineering

# Data types of columns
sdf.printSchema()

# change Income to double
sdf = sdf.withColumn("Income", sdf.Income.cast("double"))
```

```
# the feature State is a string


# StringIndexer
indexer = StringIndexer(inputCols=['State'], outputCols=['stateIndex'])


# OneHotEncoder
encoder = OneHotEncoder(inputCols=['stateIndex'], outputCols=['stateVector'], dropLast=False)


# Vector Assembler
assembler = VectorAssembler(inputCols=['stateVector','Population','Income', 'Age'],
outputCol="features")


# Create pipeline
crime_pipe = Pipeline(stages=[indexer, encoder, assembler])

# Call .fit to transform the data
transformed_sdf = crime_pipe.fit(sdf).transform(sdf)

# Review the transformed features
transformed_sdf.select('State','stateVector','Year','Population','Income','Age','Crime_per_Hundred_Thousand','aver
age_crime','highercrime','features').show(20, truncate=False)

# Split data
train, test = transformed_sdf.randomSplit([.7, .3], seed=3456)

# LogisticRegression
lr = LogisticRegression(labelCol="highercrime")

# Fit the model
model = lr.fit(train)

# Show model coefficients and intercept
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)

# Test the model on the test data
test_results = model.transform(test)

# Test Results
```

# Show the test results
test_results.select('State','Year','Population','Income','Age','Crime_per_Hundred_Thousand','average_crime','rawPrediction','probability','prediction','highercrime').show(truncate=False)

Model Validation

# Create a BinaryClassificationEvaluator to evaluate how well the model works
evaluator = evals.BinaryClassificationEvaluator(labelCol="highercrime", metricName="areaUnderROC")

AOC of test results
print(evaluator.evaluate(test_results))   # 0.9871615312791783

# Create the parameter grid (empty for now)
grid = tune.ParamGridBuilder().build()

# Create the CrossValidator
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3, seed=789)

# Use the CrossValidator to Fit the train data
cv = cv.fit(train)

Show the average performance over the three folds
cv.avgMetrics

# Evaluate the test data using the cross-validator model
# Reminder: We used Area Under the Curve
evaluator.evaluate(cv.transform(test))

Tuning
Explore a range of Hyperparameters, carry out multiple splits and then see which parameters lead to the best model performance

Create a grid to hold hyperparameters
Logistic Regression threshold from 0 to 0.1 in .01 increments

grid = tune.ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

Build the grid

```
grid = grid.build()
print('Number of models to be tested: ', len(grid))

# Create the CrossValidator using the new hyperparameter grid
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)

# Call cv.fit() to create models with all of the combinations of parameters in the grid
all_models = cv.fit(train)

print("Average Metrics for Each model: ", all_models.avgMetrics)


# Get the best model

# Gather the metrics and parameters of the model with the best average metrics
hyperparams = all_models.getEstimatorParamMaps()[np.argmax(all_models.avgMetrics)]

# Print out the list of hyperparameters for the best model
for i in range(len(hyperparams.items())):
    print([x for x in hyperparams.items()][i])

(Param(parent='LogisticRegression_daf2ca402a50', name='regParam', doc='regularization parameter (>= 0).'), 1.0)
(Param(parent='LogisticRegression_daf2ca402a50', name='elasticNetParam', doc='the ElasticNet mixing parameter, in
range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.'), 0.0)

# Choose the best model
bestModel = all_models.bestModel




# Visualizations


# Show the confusion matrix
test_results.groupby('highercrime').pivot('prediction').count().show()


cm = test_results.groupby('highercrime').pivot('prediction').count().fillna(0).collect()

def calculate_precision_recall(cm):
    tn = cm[0][1]
    fp = cm[0][2]
    fn = cm[1][1]
```

```
    tp = cm[1][2]
    precision = tp / ( tp + fp )
    recall = tp / ( tp + fn )
    accuracy = ( tp + tn ) / ( tp + tn + fp + fn )
    f1_score = 2 * ( ( precision * recall ) / ( precision + recall ) )
    return accuracy, precision, recall, f1_score

print(calculate_precision_recall(cm))    # (0.04580152671755725, 0.05714285714285714, 0.06349206349206349,
0.06015037593984963)




# ROC CURVE

plt.figure(figsize=(6,6))
plt.plot([0, 1], [0, 1], 'r--')
x = bestModel.summary.roc.select('FPR').collect()
y = bestModel.summary.roc.select('TPR').collect()
plt.scatter(x, y)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")

# Save plot to S3
# Create a buffer to hold the figure
img_data = io.BytesIO()

# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)

# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/ROC_Curve.png', 'wb') as f:
    f.write(img_data.getbuffer())




# Coefficients

# Extract the coefficients on each of the variables
coeff = bestModel.coefficients.toArray().tolist()
```

```
# Loop through the features to extract the original column names. Store in the var_index dictionary
var_index = dict()
for variable_type in ['numeric', 'binary']:
    for variable in test_results.schema["features"].metadata["ml_attr"]["attrs"][variable_type]:
        print("Found variable:", variable)
        idx = variable['idx']
        name = variable['name']
        var_index[idx] = name # Add the name to the dictionary

# Loop through all of the variables found and print out the associated coefficients
for i in range(len(var_index)):
    print(i, var_index[i], coeff[i])
```

```
# Convert national average (from groupby) spark DataFrame to pandas DataFrame for line graphs
df2 = average_crime_sdf.toPandas()
```

```
# Line graph of change in national average crime by year

# Have to sort the Year column in order to have the correct line graph
df2_sorted = df2.sort_values('Year')

# Create line graph
plt.plot(df2_sorted['Year'], df2_sorted['average_crime'], linestyle='--', color='red', marker='o',label='Average
National Crime')

# add title and labels
plt.title('United States Average National Crime by Year')
plt.xlabel('Year')
plt.ylabel('Average National Crime')

# add gridlines
plt.grid(True)

# Save plot to S3
# Create a buffer to hold the figure
img_data = io.BytesIO()

# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)
```

```
# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/line_national_crime.png', 'wb') as f:
    f.write(img_data.getbuffer())






# Bar graph combined with line graph of percent change in national average crime by year

# Have to sort the Year column in order to have the correct line graph
df2_sorted = df2.sort_values('Year')


# Calculate the percent change of crime with each increasing year
df2_sorted['crime_pct_change'] = df2_sorted['average_crime'].pct_change()


# Remove the first value (nan) of 2011 because there is no previous year in the dataset
df2_sorted = df2_sorted.dropna()


# Create line graph
plt.plot(df2_sorted['Year'], df2_sorted['crime_pct_change'], linestyle='--', color='red', marker='o',label='Percent
Change of Average National Crime')

# Create bar graph
plt.bar(x=df2_sorted['Year'], height=df2_sorted['crime_pct_change'], width=0.4, color='blue', edgecolor='k')


# Set the y-axis tick labels to be shown as percentages with the percent symbol (multiplies the number by 100 and
adds % symbol)
plt.gca().yaxis.set_major_formatter(PercentFormatter(xmax=1.0))

# Invert the y-axis so the bars appear in the correct orientation
plt.gca().invert_yaxis()


# add title and labels
plt.title('United States Percent Change in Average National Crime by Year')
plt.xlabel('Year')
plt.ylabel('Percent Change in Average National Crime')

# add gridlines
```

```
plt.grid(True)

# Save plot to S3
# Create a buffer to hold the figure
img_data = io.BytesIO()

# Write the figure to the buffer
plt.savefig(img_data, format='png', bbox_inches='tight')
img_data.seek(0)

# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/line_bar_pct_change_national_crime.png', 'wb') as f:
  f.write(img_data.getbuffer())
```

```
# Convert spark DataFrame to pandas DataFrame for plotly visuals
df = sdf.toPandas()
```

```
# Map of U.S. for whether the average crime rate per 100,000 people for each state is greater than the national average for 2011

# create DataFrame filtering values only from 2011
df2 = df[df.Year == 2011]     # filter the original DataFrame to only contain rows where Year column is 2011
df2 = df2.rename({'State Crime': 'State Crime 2011'}, axis=1)

# Create map
fig = px.choropleth(df2,
            locations='State_Abbreviation',
            locationmode="USA-states",
            scope="usa",
            color='State Crime 2011',
            color_continuous_scale="viridis_r"
            )

# Add titles
fig.update_layout(
    title_text = '2011 Crime Greater Than National Average by State',
    title_font_family="Times New Roman",
    title_font_size = 22,
    title_font_color="black",
```

```
    title_x=0.45,
      )


# Save plot to S3
# Create a buffer to hold the figure
img_data = io.BytesIO()

# Write the figure to the buffer
fig.write_image(img_data)
img_data.seek(0)

# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/map_national_crime_2011.png', 'wb') as f:
  f.write(img_data.getbuffer())




# Map of U.S. for whether the average crime rate per 100,000 people for each state is greater than the national
average for 2019

# create DataFrame filtering values only from 2019
df3 = df[df.Year == 2019]     # filter the original DataFrame to only contain rows where Year column is 2019
df3 = df3.rename({'State Crime': 'State Crime 2019'}, axis=1)

# Create map
fig = px.choropleth(df3,
              locations='State_Abbreviation',
              locationmode="USA-states",
              scope="usa",
              color='State Crime 2019',
              color_continuous_scale="viridis_r"
              )

# Add titles
fig.update_layout(
    title_text = '2019 Crime Greater Than National Average by State',
    title_font_family="Times New Roman",
    title_font_size = 22,
    title_font_color="black",
    title_x=0.45,
      )
```

```
# Save plot to S3
# Create a buffer to hold the figure
img_data = io.BytesIO()

# Write the figure to the buffer
fig.write_image(img_data)
img_data.seek(0)

# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/map_national_crime_2019.png', 'wb') as f:
   f.write(img_data.getbuffer())
```

```
# Map of U.S. for the difference in Crime_Per_Hundred_Thousand and average_crime (national average crime per 100,000) in 2011

# create DataFrame filtering values only from 2011
df4 = df[df.Year == 2011]     # filter the original DataFrame to only contain rows where Year column is 2011
df4['Crime Difference 2011'] = df4['Crime_Per_Hundred_Thousand'] - df4['average_crime']


# Create map
fig = px.choropleth(df4,
                locations='State_Abbreviation',
                locationmode="USA-states",
                scope="usa",
                color='Crime Difference 2011',
                color_continuous_scale="blues"
                )

# Add titles
fig.update_layout(
    title_text = '2011 Difference Between Crime and National Average by State',
    title_font_family="Times New Roman",
    title_font_size = 19,
    title_font_color="black",
    title_x=0.45,
      )


# Save plot to S3
```

```
# Create a buffer to hold the figure
img_data = io.BytesIO()

# Write the figure to the buffer
fig.write_image(img_data)
img_data.seek(0)

# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/map_crime_difference_2011.png', 'wb') as f:
  f.write(img_data.getbuffer())




# Map of U.S. for the difference in Crime_Per_Hundred_Thousand and average_crime (national average crime per 100,000) in 2019

# create DataFrame filtering values only from 2019
df5 = df[df.Year == 2019]    # filter the original DataFrame to only contain rows where Year column is 2019
df5['Crime Difference 2019'] = df5['Crime_Per_Hundred_Thousand'] - df5['average_crime']


# Create map
fig = px.choropleth(df5,
            locations='State_Abbreviation',
            locationmode="USA-states",
            scope="usa",
            color='Crime Difference 2019',
            color_continuous_scale="blues"
            )

# Add titles
fig.update_layout(
    title_text = '2019 Difference Between Crime and National Average by State',
    title_font_family="Times New Roman",
    title_font_size = 19,
    title_font_color="black",
    title_x=0.45,
     )


# Save plot to S3
# Create a buffer to hold the figure
img_data = io.BytesIO()
```

```
# Write the figure to the buffer
fig.write_image(img_data)
img_data.seek(0)

# Connect to the s3fs file system
s3 = s3fs.S3FileSystem(anon=False)
with s3.open('s3://big-data-project-1/map_crime_difference_2019.png', 'wb') as f:
    f.write(img_data.getbuffer())
```

Appendix D

Complete Data Processing Pipeline

Notes:

- The code may paste in a way that causes an error so double check it

```
# create column for crime per 100,000 people
sdf = sdf.withColumn('Crime_Per_Hundred_Thousand', floor(sdf.Crime / sdf.Population * 100000))
# calculate national average number of crimes per 100,000 people for each year
average_crime_sdf = sdf.groupBy('Year').agg(round(mean('Crime_Per_Hundred_Thousand')).alias('average_crime'))
# Join the average crime DataFrame back into the original DataFrame sdf
sdf = sdf.join(average_crime_sdf, "Year")
# Create the label, =1 if Crime_per_Hundred_Thousand > average_crime, =0 otherwise
sdf = sdf.withColumn("highercrime", when(sdf.Crime_Per_Hundred_Thousand > sdf.average_crime,
1.0).otherwise(0.0))
# change Income to double
sdf = sdf.withColumn("Income", sdf.Income.cast("double"))
# StringIndexer
indexer = StringIndexer(inputCols=['State'], outputCols=['stateIndex'])
# OneHotEncoder
encoder = OneHotEncoder(inputCols=['stateIndex'], outputCols=['stateVector'], dropLast=False)
# Vector Assembler
assembler = VectorAssembler(inputCols=['stateVector','Population','Income', 'Age'],
outputCol="features")
# Create pipeline
crime_pipe = Pipeline(stages=[indexer, encoder, assembler])
# Call .fit to transform the data
transformed_sdf = crime_pipe.fit(sdf).transform(sdf)
# Split data
train, test = transformed_sdf.randomSplit([.7, .3], seed=3456)
# LogisticRegression
lr = LogisticRegression(labelCol="highercrime")
# Fit the model
model = lr.fit(train)
# Test the model on the test data
test_results = model.transform(test)
# Create a BinaryClassificationEvaluator to evaluate how well the model works
evaluator = evals.BinaryClassificationEvaluator(labelCol="highercrime", metricName="areaUnderROC")
# Create the parameter grid (empty for now)
grid = tune.ParamGridBuilder().build()
# Create the CrossValidator
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3, seed=789)
# Use the CrossValidator to Fit the train data
cv = cv.fit(train)
# Evaluate the test data using the cross-validator model
```

```
evaluator.evaluate(cv.transform(test))
# Create a grid to hold hyperparameters
grid = tune.ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])
# Build the grid
grid = grid.build()
# Create the CrossValidator using the new hyperparameter grid
cv = tune.CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)
# Call cv.fit() to create models with all of the combinations of parameters in the grid
all_models = cv.fit(train)
# Choose the best model
bestModel = all_models.bestModel
# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test)
# Evaluate the predictions. Area Under ROC curve
pred = evaluator.evaluate(test_results)
```