Introduction to Big Data and Data Science

Project Document

Group 11

Topic:

Big Data Science on COVID-19 Data

Team Members:

Gopi Srinivas Yerramothu

11654173

Sreelekha Onteddu

11661665

Madhuri Addla

11594379

Sarika Kandey

11604330

Indira Devi Siripurapu

11550109

Abstract:

In the contemporary big data era, large volumes of big data may be created and gathered quickly from a wide range of rich data sources. This huge data contains significant knowledge and practical information. Examples include information on healthcare and epidemiology, such as information on individuals with viral infections like the coronavirus disease in 2019. (COVID-19). Data scientists' knowledge gained from these epidemiological data helps researchers, epidemiologists, and policymakers better understand the disease, which may motivate them to devise strategies for detecting, containing, and battling it. This article offers a data science approach to analyzing extensive COVID-19 epidemiological data.

Users can better grasp information regarding COVID-19 confirmed cases thanks to the solution. The advantages of our data science solution in extracting knowledge from the massive COVID-19 data are demonstrated by evaluation results.

Input dataset:

Gist link of the dataset:

https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv

This Dataset is a mixture of many datasets found on Kaggle. We collaborated on many datasets according to the project requirements.

The dataset is about COVID-19, the pandemic virus. We have 2000 rows of data, and these columns:

submission_date,tot_cases,conf_cases,prob_cases,new_case,pnew_case,tot_death,conf_death,prob_death,new_death,pnew_death,created_at,consent_cases,consent_deaths,Gender,State,Population,Age.

Introduction:

Our research aims to develop a comprehensive data analysis solution for processing and interpreting large sets of COVID-19 epidemiological data. Our study offers significant contributions to data science, specifically by providing a range of functionalities for conducting an in-depth analysis of COVID-19 data. These functionalities enable researchers to uncover significant trends and insights, such as identifying patterns in disease spread, tracking the efficacy of public health interventions, and predicting future outbreaks. Our solution represents a valuable tool for researchers and policymakers to better understand and combat the ongoing COVID-19 pandemic.

Description of the Model Approach:

Frequent Pattern Mining:

A "frequent pattern mining" data mining approach seeks to find recurrent patterns or connections in a dataset. Frequent patterns can be found using algorithms like Apriori, FP-Growth, and Eclat. Analyzing market baskets, recommendation systems, and DNA sequences are just a few of the applications where these patterns are helpful.

Contrast Pattern Mining

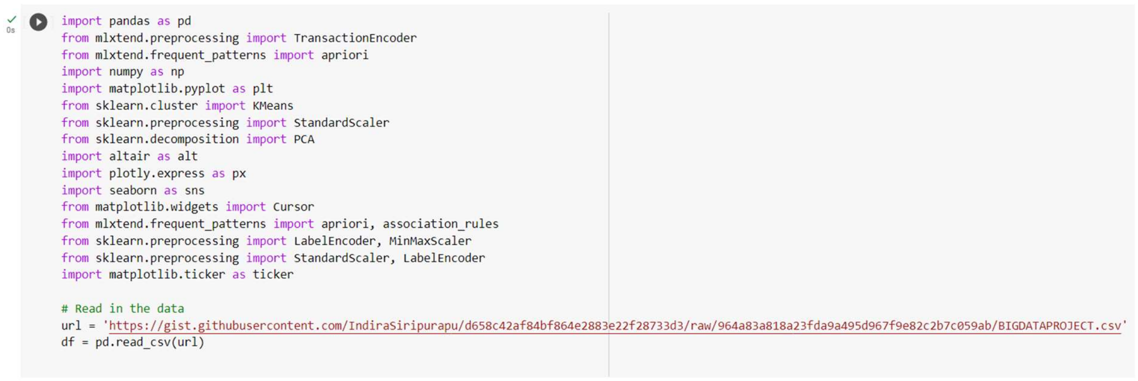
Contrast Pattern Mining is a subset of Frequent Pattern Mining that looks for common patterns in one data set but uncommon in another. The difference between the behaviors of two groups of customers, for example, can be determined by using this technique to compare two datasets side by side.

K Means Clustering:

K Means Clustering is an unsupervised machine learning method that combines related data points. The process begins by randomly picking K centroids, where K is the required number of clusters. It then determines a new centroid for each cluster and assigns each data point to the closest centroid. The process is repeated until the centroids stop moving or a predetermined number of iterations has been reached. Applications like data mining, image processing, and natural language processing frequently use K Means Clustering.

Code screenshots, Outputs and Analysis:

Pre-processing:

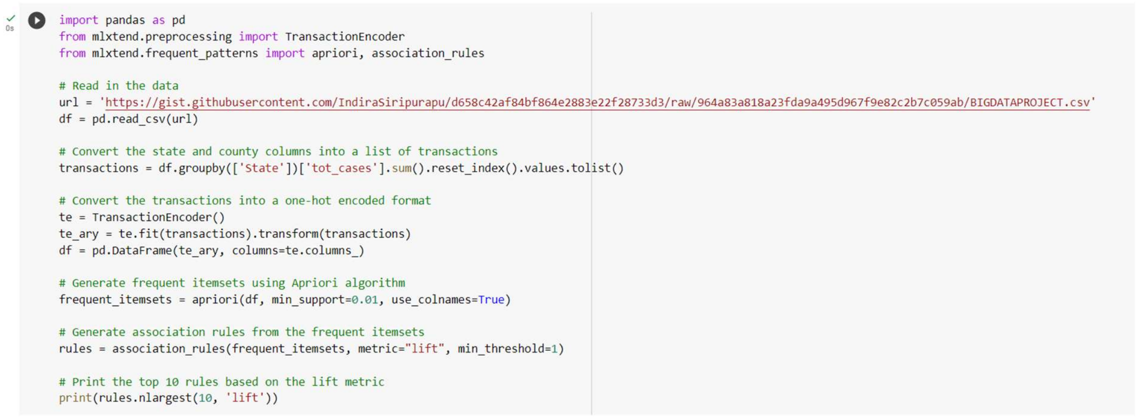


```python
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import altair as alt
import plotly.express as px
import seaborn as sns
from matplotlib.widgets import Cursor
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.preprocessing import StandardScaler, LabelEncoder
import matplotlib.ticker as ticker

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)
```

Analysis:

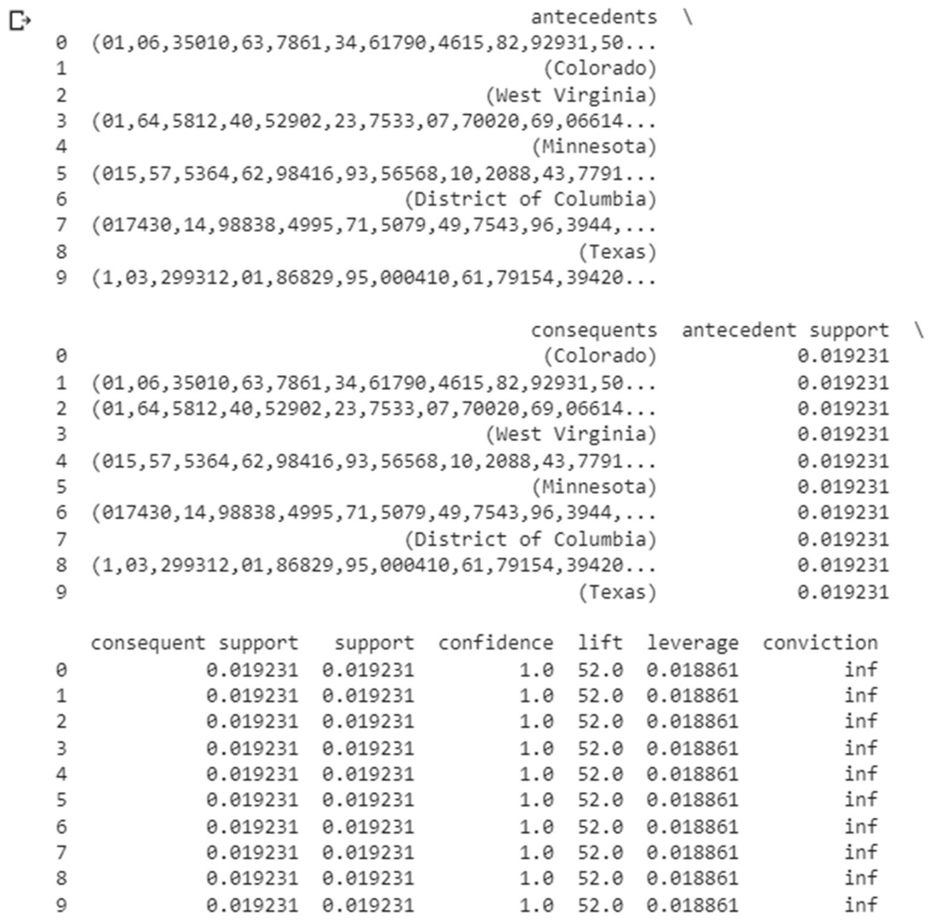
In this code, we have used several Python libraries to analyze and visualize the BIGDATAPPROJECT dataset. The dataset is read from a CSV file on GitHub. The libraries used are Pandas, mlxtend, numPy, matplotlib, seaborn, altair, plotly express, scikit-learn, etc. This code aims to explore and analyze the dataset with specific tasks such as frequent pattern mining, clustering, feature scaling, principal component analysis, etc. The code also shows how to use different visualization libraries such as Matplotlib, Seq, Altair, and Plotly Express for static and interactive visualization.

Code:



```python
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Convert the state and county columns into a list of transactions
transactions = df.groupby(['State'])['tot_cases'].sum().reset_index().values.tolist()

# Convert the transactions into a one-hot encoded format
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Generate frequent itemsets using Apriori algorithm
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

# Generate association rules from the frequent itemsets
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Print the top 10 rules based on the lift metric
print(rules.nlargest(10, 'lift'))
```

Output:

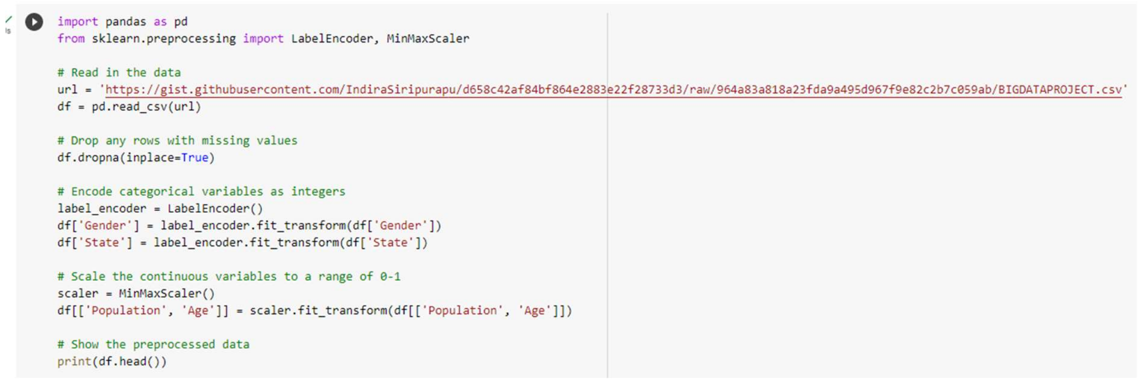


```
                                    antecedents  \
0  (01,06,35010,63,7861,34,61790,4615,82,92931,50...
1                                        (Colorado)
2                                   (West Virginia)
3  (01,64,5812,40,52902,23,7533,07,70020,69,06614...
4                                       (Minnesota)
5  (015,57,5364,62,98416,93,56568,10,2088,43,7791...
6                            (District of Columbia)
7  (017430,14,98838,4995,71,5079,49,7543,96,3944,...
8                                           (Texas)
9  (1,03,299312,01,86829,95,000410,61,79154,39420...

                                    consequents  antecedent support  \
0                                        (Colorado)            0.019231
1  (01,06,35010,63,7861,34,61790,4615,82,92931,50...            0.019231
2  (01,64,5812,40,52902,23,7533,07,70020,69,06614...            0.019231
3                                   (West Virginia)            0.019231
4  (015,57,5364,62,98416,93,56568,10,2088,43,7791...            0.019231
5                                       (Minnesota)            0.019231
6  (017430,14,98838,4995,71,5079,49,7543,96,3944,...            0.019231
7                            (District of Columbia)            0.019231
8  (1,03,299312,01,86829,95,000410,61,79154,39420...            0.019231
9                                           (Texas)            0.019231

   consequent support   support  confidence  lift  leverage  conviction
0            0.019231  0.019231         1.0  52.0  0.018861         inf
1            0.019231  0.019231         1.0  52.0  0.018861         inf
2            0.019231  0.019231         1.0  52.0  0.018861         inf
3            0.019231  0.019231         1.0  52.0  0.018861         inf
4            0.019231  0.019231         1.0  52.0  0.018861         inf
5            0.019231  0.019231         1.0  52.0  0.018861         inf
6            0.019231  0.019231         1.0  52.0  0.018861         inf
7            0.019231  0.019231         1.0  52.0  0.018861         inf
8            0.019231  0.019231         1.0  52.0  0.018861         inf
9            0.019231  0.019231         1.0  52.0  0.018861         inf
```

Analysis:

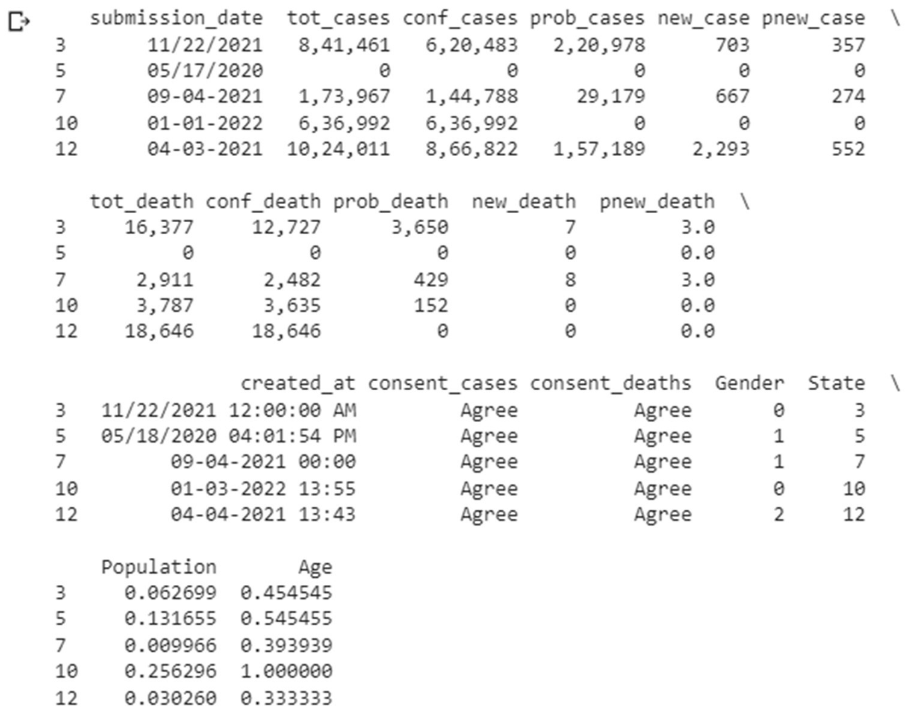
The apriori algorithm is utilized in this code to analyze the BigDataAProject dataset and identify frequent itemsets and associated rules between the State column and tot_cases column. The dataset is read from a CSV file hosted on GitHub, converted into a list of transactions, and then encoded into a one-hot encoded format with the UseMlxtend TransactionEncoder class. The apriori function is used to generate frequent item sets, and the association rules are generated from frequent item sets with the AssociationRules function. Rules are evaluated using the lift metric and a minimum threshold of 1, with the top 10 rules being printed based on lift.

Code:



```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Drop any rows with missing values
df.dropna(inplace=True)

# Encode categorical variables as integers
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['State'] = label_encoder.fit_transform(df['State'])

# Scale the continuous variables to a range of 0-1
scaler = MinMaxScaler()
df[['Population', 'Age']] = scaler.fit_transform(df[['Population', 'Age']])

# Show the preprocessed data
print(df.head())
```

Output:

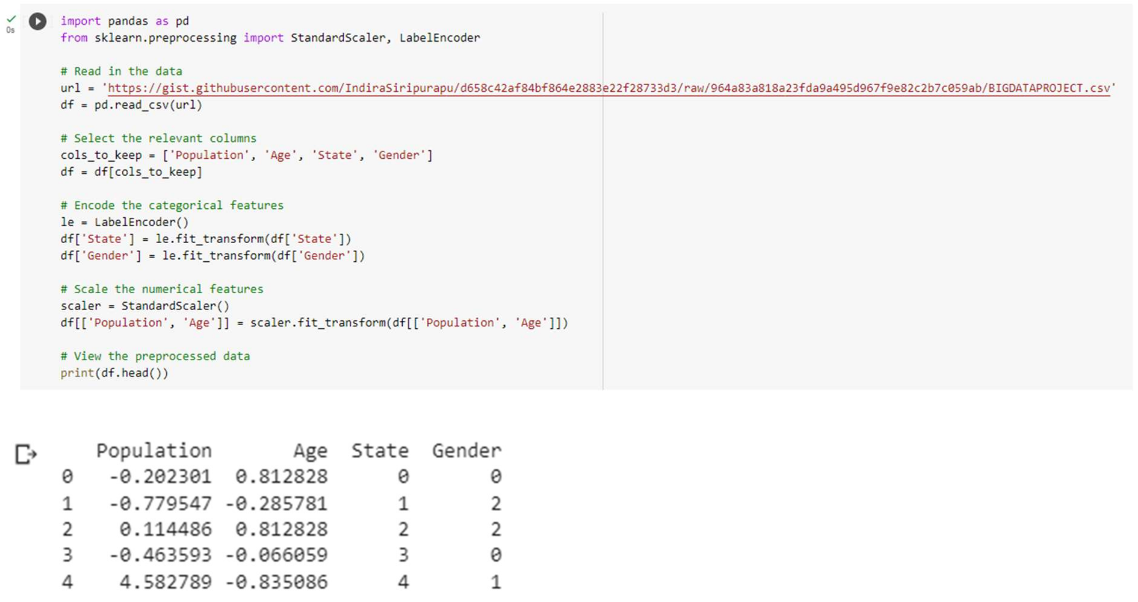


```
   submission_date  tot_cases conf_cases prob_cases new_case pnew_case  \
3       11/22/2021   8,41,461   6,20,483   2,20,978      703       357
5       05/17/2020          0          0          0        0         0
7       09-04-2021   1,73,967   1,44,788     29,179      667       274
10      01-01-2022   6,36,992   6,36,992          0        0         0
12      04-03-2021  10,24,011   8,66,822   1,57,189    2,293       552

   tot_death conf_death prob_death  new_death  pnew_death  \
3     16,377     12,727      3,650          7         3.0
5          0          0          0          0         0.0
7      2,911      2,482        429          8         3.0
10     3,787      3,635        152          0         0.0
12    18,646     18,646          0          0         0.0

                created_at consent_cases consent_deaths  Gender  State  \
3    11/22/2021 12:00:00 AM         Agree          Agree       0      3
5    05/18/2020 04:01:54 PM         Agree          Agree       1      5
7          09-04-2021 00:00         Agree          Agree       1      7
10         01-03-2022 13:55         Agree          Agree       0     10
12         04-04-2021 13:43         Agree          Agree       2     12

    Population       Age
3     0.062699  0.454545
5     0.131655  0.545455
7     0.009966  0.393939
10    0.256296  1.000000
12    0.030260  0.333333
```

Analysis:

In this code, we are going to clean and transform a BigData dataset for analysis. We are going to read the BigData dataset from a CSV file on GitHub. Pandas is going to be used to read the BigData dataset from the CSV file. We will drop the rows that contain missing values in the BigData dataset. In the BigData dataset, the categorical columns ‘Gender’ and ‘State’ are encoded as integers using the label encoder class (Scikit-learn).The continuous columns (population) and (age) are scaled (MinMaxScaler) from 0 to 1. Using the head method in Pandas, we print the first few lines of the transformed dataset.

Code:



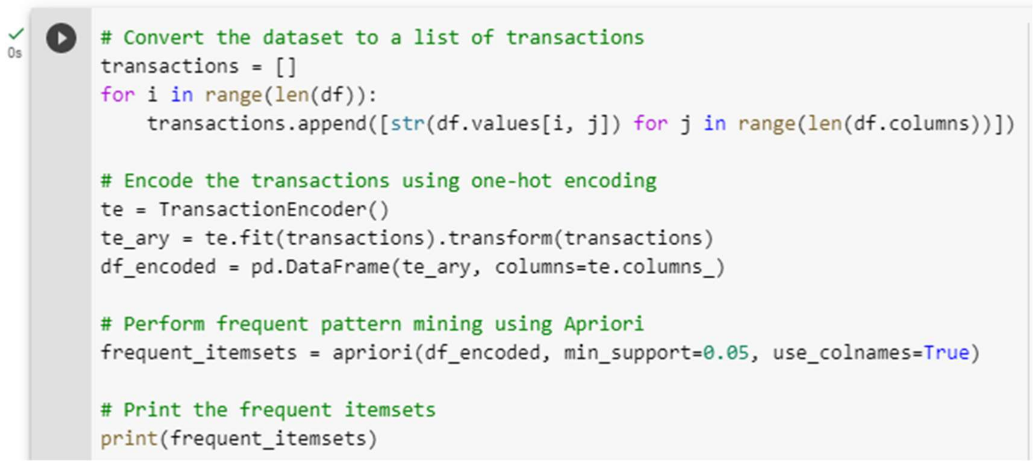
```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Select the relevant columns
cols_to_keep = ['Population', 'Age', 'State', 'Gender']
df = df[cols_to_keep]

# Encode the categorical features
le = LabelEncoder()
df['State'] = le.fit_transform(df['State'])
df['Gender'] = le.fit_transform(df['Gender'])

# Scale the numerical features
scaler = StandardScaler()
df[['Population', 'Age']] = scaler.fit_transform(df[['Population', 'Age']])

# View the preprocessed data
print(df.head())
```

```
   Population       Age  State  Gender
0   -0.202301  0.812828      0       0
1   -0.779547 -0.285781      1       2
2    0.114486  0.812828      2       2
3   -0.463593 -0.066059      3       0
4    4.582789 -0.835086      4       1
```

Analysis:

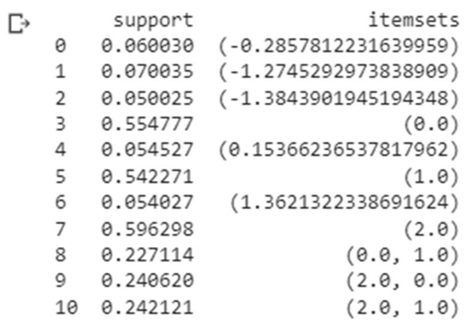
In this code, we will preprocess the big data dataset to make it ready for analysis. We will read the big data dataset from the CSV file on GitHub using pandas. Only the important columns (population, age, state, and gender) will be kept in the big data dataset using the list indexing method. The categorical features (state, gender) will be encoded as an integer using the label encoder class (Scikit-Learn) from the big data library. The numerical features (population, age) will be standardized to have an average (0) and standard deviation (1) using the standard scaler class (Sikkit-learn). The preprocessed data will be printed with the head method (Pandas) to display the first few lines.

Frequent Pattern Mining:



```
# Convert the dataset to a list of transactions
transactions = []
for i in range(len(df)):
    transactions.append([str(df.values[i, j]) for j in range(len(df.columns))])

# Encode the transactions using one-hot encoding
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Perform frequent pattern mining using Apriori
frequent_itemsets = apriori(df_encoded, min_support=0.05, use_colnames=True)

# Print the frequent itemsets
print(frequent_itemsets)
```

Output:

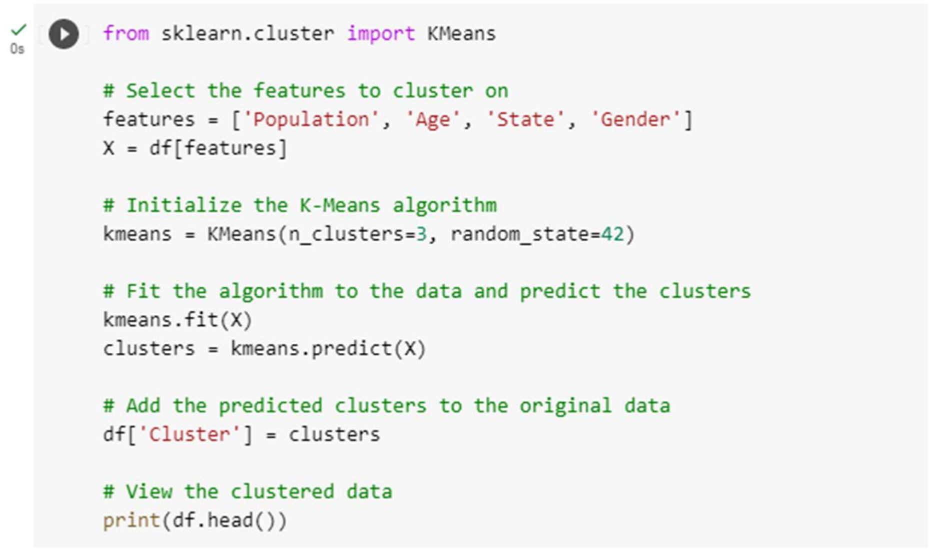


```
     support              itemsets
0   0.060030  (-0.2857812231639959)
1   0.070035  (-1.2745292973838909)
2   0.050025  (-1.3843901945194348)
3   0.554777                 (0.0)
4   0.054527  (0.15366236537817962)
5   0.542271                 (1.0)
6   0.054027   (1.3621322338691624)
7   0.596298                 (2.0)
8   0.227114            (0.0, 1.0)
9   0.240620            (2.0, 0.0)
10  0.242121            (2.0, 1.0)
```

Analysis:

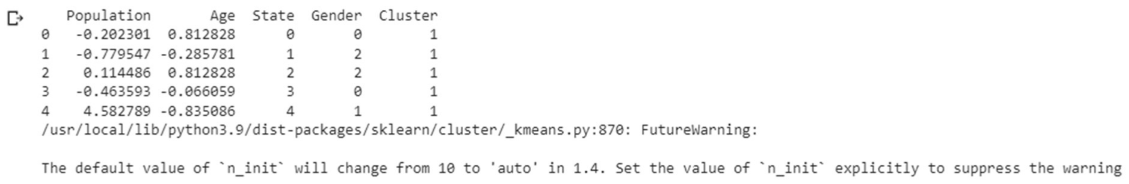
Frequently, pattern mining is a method of identifying patterns or relationships between items in a set of data. It is commonly employed in data mining and machine learning to detect associations or correlations between items in the data, which can be used to form predictions or gain insight into the data. Apriori is a classic frequent pattern mining algorithm, which works by identifying sets of items that occur frequently in the dataset. The algorithm utilizes a support threshold, which is the minimum frequency that an itemset must have in order to be classified as "frequent". The dataset is first converted into a list of transactions, a common format used for frequent pattern mining . Each transaction represents a record in the data set and contains a list of values per attribute. One-hot encoding is then used to encode the transactions using binary vectors. After the transactions have been encoded, Apriori is applied using the MLxtend library, and the frequent item sets resulting from this algorithm are printed out to the console.

K Means Clustering:



```python
from sklearn.cluster import KMeans

# Select the features to cluster on
features = ['Population', 'Age', 'State', 'Gender']
X = df[features]

# Initialize the K-Means algorithm
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the algorithm to the data and predict the clusters
kmeans.fit(X)
clusters = kmeans.predict(X)

# Add the predicted clusters to the original data
df['Cluster'] = clusters

# View the clustered data
print(df.head())
```

Output:

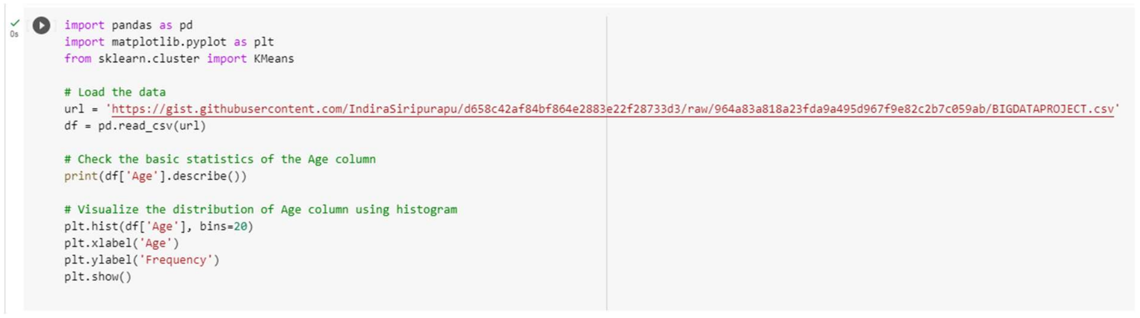


```
   Population       Age  State  Gender  Cluster
0   -0.202301  0.812828      0       0        1
1   -0.779547 -0.285781      1       2        1
2    0.114486  0.812828      2       2        1
3   -0.463593 -0.066059      3       0        1
4    4.582789 -0.835086      4       1        1
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

Analysis:

In this code, we are going to use the K-means algorithm with the population, age, state, and gender columns as the features for clustering. We will initialize the algorithm with three clusters and random state 42. Then, we will call the fit method with the algorithm and the feature data. Finally, we call the predict method with the predicted cluster labels. We will add the predicted cluster labels to our original dataset as the new column 'Cluster'. Finally, we will print the resulting dataset using Pandas' head method to display the first few lines.

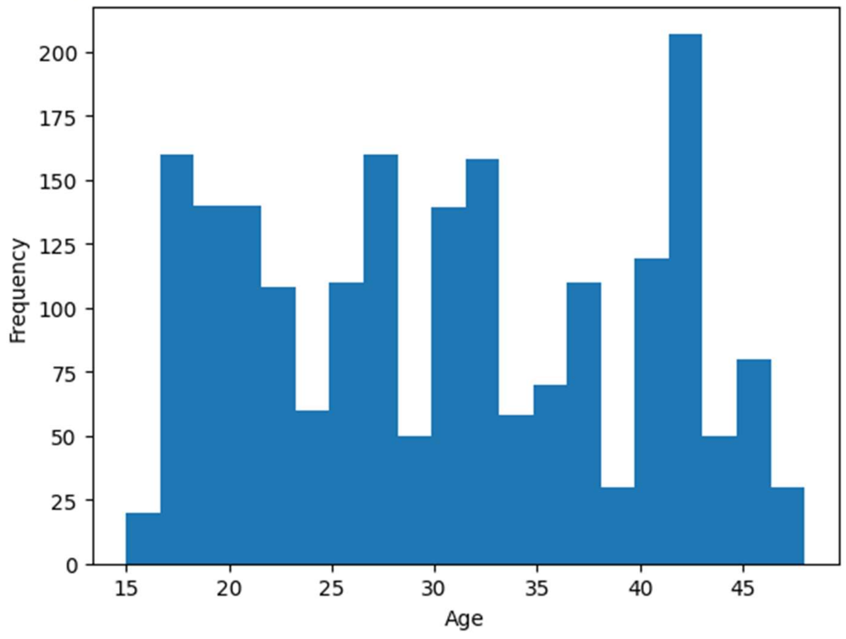
Code:



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Check the basic statistics of the Age column
print(df['Age'].describe())

# Visualize the distribution of Age column using histogram
plt.hist(df['Age'], bins=20)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

Output:

```
count    1999.000000
mean       30.601301
std         9.104697
min        15.000000
25%        22.000000
50%        30.000000
75%        39.000000
max        48.000000
Name: Age, dtype: float64
```

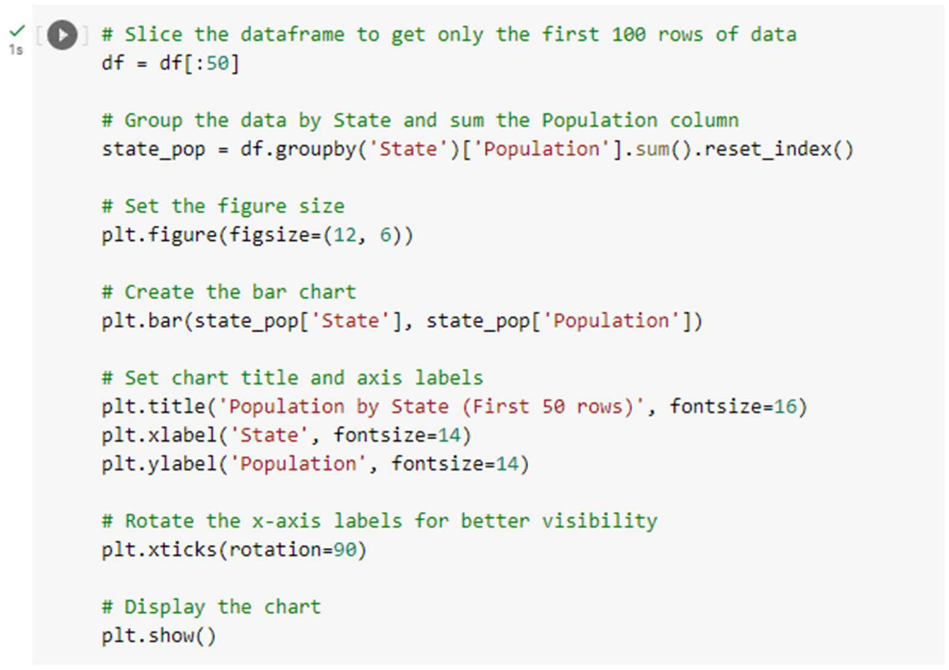


Analysis:

In this code, we are going to load a data set from a URL, and then we are going to generate a histogram that shows the distribution of that particular column in the data set. We are also going to print out some basic statistics for that particular column using the pdas.describe() function. We are going to load the data set using pdas.read_csv() and then use pdcs.describe() to get the basic stats for that particular Age column. Then, we will generate a histogram using the pdcs.hist() function and then use the pdmas.pyplot() function to show the histogram. In summary, this code will help you understand the distribution of that specific column in your data set, which can help you understand the characteristics of your data.

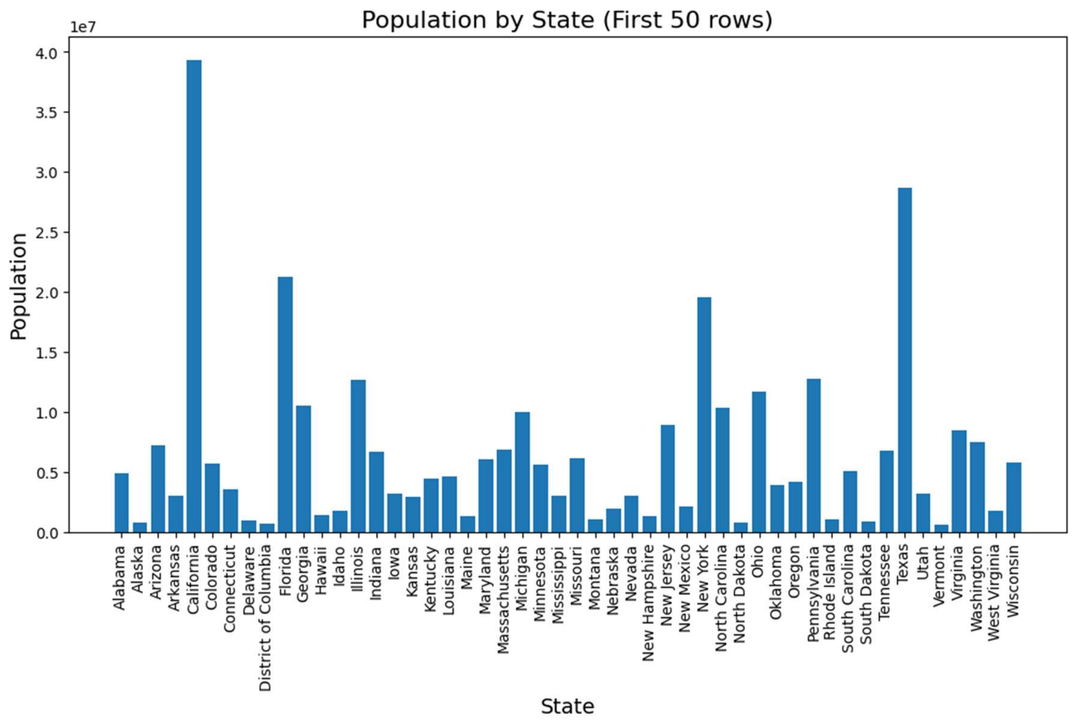
Visualizations:

Code:



```
# Slice the dataframe to get only the first 100 rows of data
df = df[:50]

# Group the data by State and sum the Population column
state_pop = df.groupby('State')['Population'].sum().reset_index()

# Set the figure size
plt.figure(figsize=(12, 6))

# Create the bar chart
plt.bar(state_pop['State'], state_pop['Population'])

# Set chart title and axis labels
plt.title('Population by State (First 50 rows)', fontsize=16)
plt.xlabel('State', fontsize=14)
plt.ylabel('Population', fontsize=14)

# Rotate the x-axis labels for better visibility
plt.xticks(rotation=90)

# Display the chart
plt.show()
```

Output:

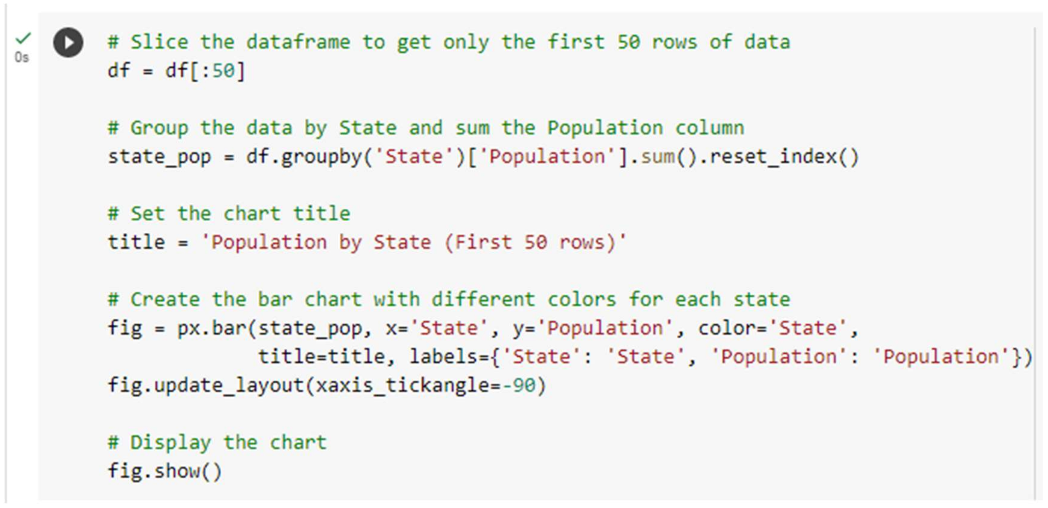


Analysis:

The code provided loads data into a Pandas DataFrame from a CSV file stored at the URL ,The essential statistics of the 'Age' column are then printed using Pandas DataFrame's describe() method.

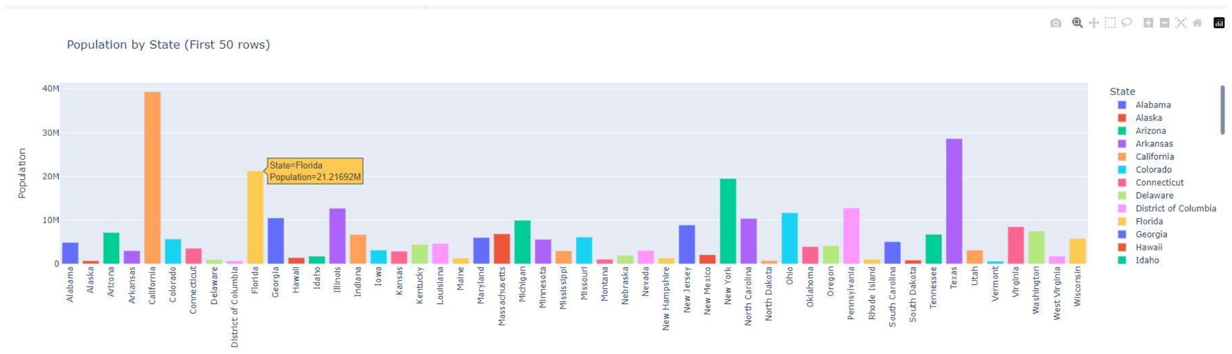
The distribution of the 'Age' column can be seen using a histogram produced with Matplotlib. Using the 'bins' argument of the 'plt.hist()' function, the 'Age' values have been divided into 20 bins. The x-axis label is "Age," while the y-axis label is "Frequency." Finally, using the 'plt.show()' method, the histogram is presented.

Code:



```
# Slice the dataframe to get only the first 50 rows of data
df = df[:50]

# Group the data by State and sum the Population column
state_pop = df.groupby('State')['Population'].sum().reset_index()

# Set the chart title
title = 'Population by State (First 50 rows)'

# Create the bar chart with different colors for each state
fig = px.bar(state_pop, x='State', y='Population', color='State',
             title=title, labels={'State': 'State', 'Population': 'Population'})
fig.update_layout(xaxis_tickangle=-90)

# Display the chart
fig.show()
```

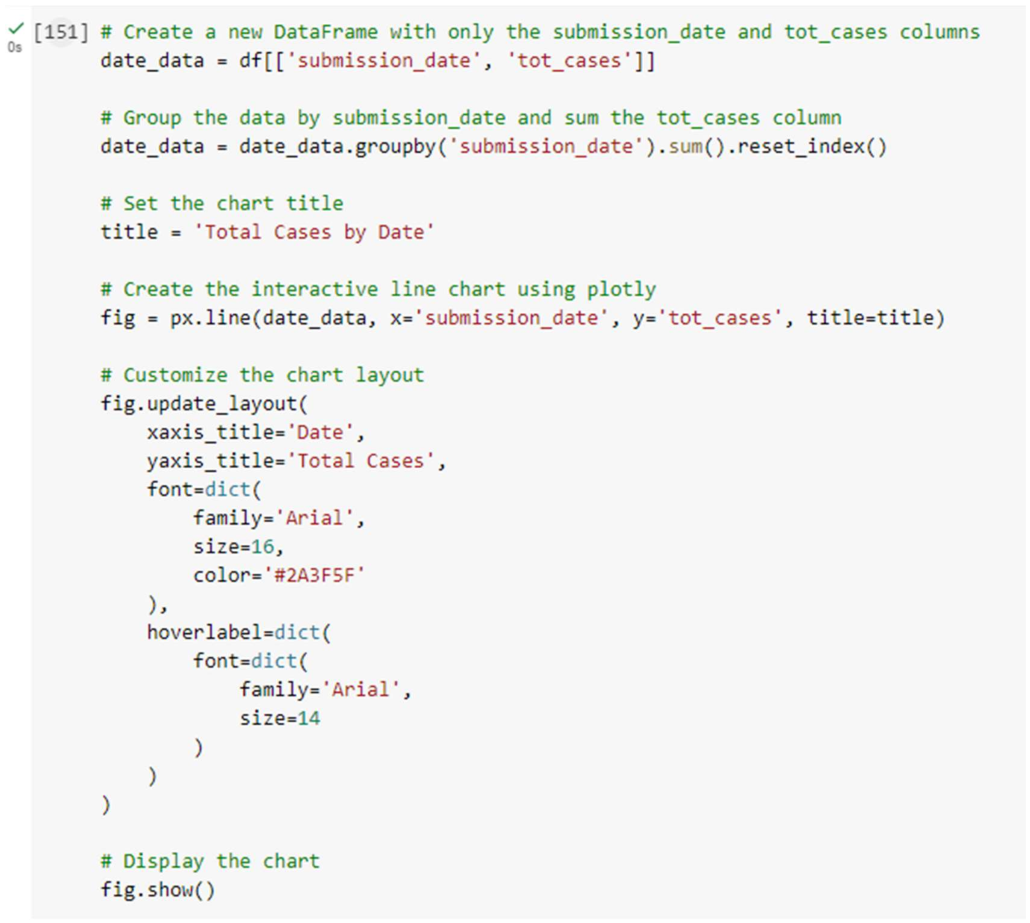
Output:



Analysis:

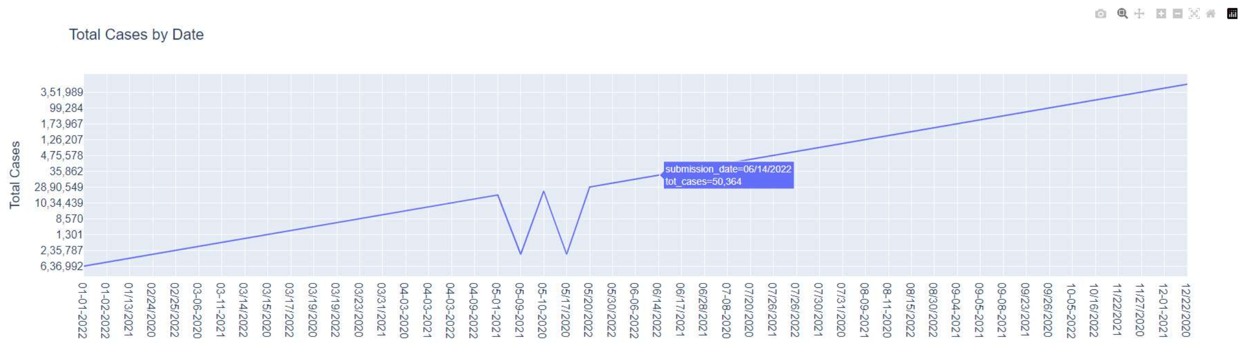
The code slices DataFrame df to get only the first 50 rows, groups the data by state, calculates the sum of the population column for each state, and creates a bar chart using Matplotlib with the x-axis representing the states and the y-axis representing the population.The chart title is set to 'Population by State (First 50 rows)', the x-axis and y-axis labels are set to 'State' and 'Population', and the chart is displayed using the plt.show() function

Code:



```
[151] # Create a new DataFrame with only the submission_date and tot_cases columns
      date_data = df[['submission_date', 'tot_cases']]

      # Group the data by submission_date and sum the tot_cases column
      date_data = date_data.groupby('submission_date').sum().reset_index()

      # Set the chart title
      title = 'Total Cases by Date'

      # Create the interactive line chart using plotly
      fig = px.line(date_data, x='submission_date', y='tot_cases', title=title)

      # Customize the chart layout
      fig.update_layout(
          xaxis_title='Date',
          yaxis_title='Total Cases',
          font=dict(
              family='Arial',
              size=16,
              color='#2A3F5F'
          ),
          hoverlabel=dict(
              font=dict(
                  family='Arial',
                  size=14
              )
          )
      )

      # Display the chart
      fig.show()
```

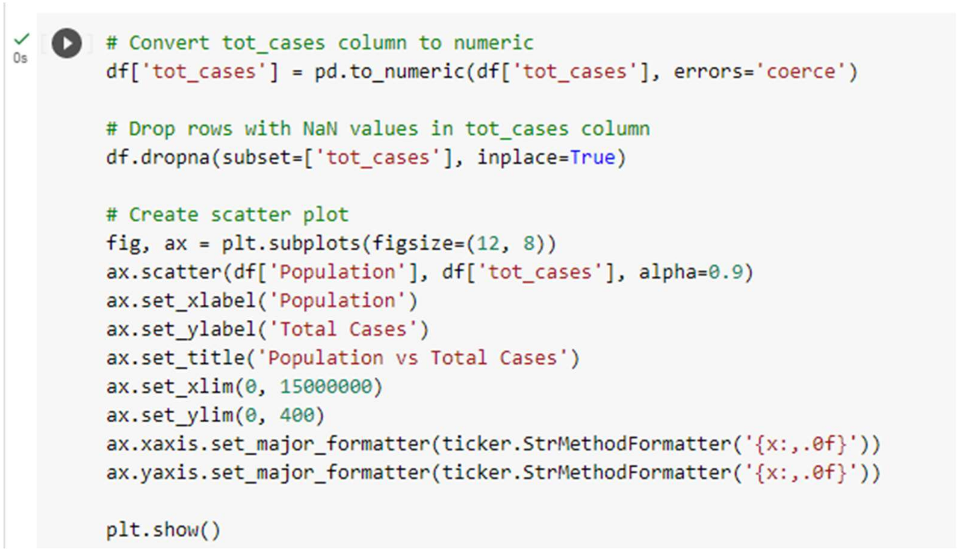
Output:



Analysis:

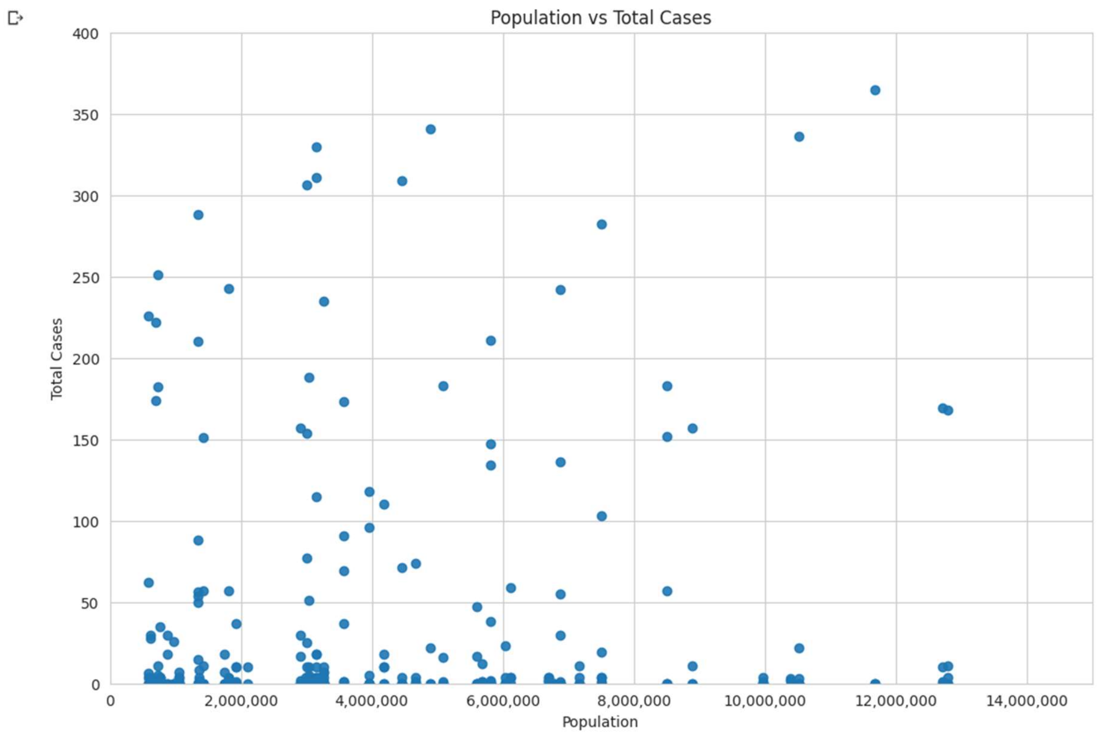
The code uses the plotly library to create a bar chart with different colors for each state. The data is grouped by State and sum of Population column, and the chart title is set using the 'title' variable. The bar chart is created with x-axis representing the 'State' column, y-axis representing the 'Population' column, and color-coded by 'State'. The x-axis labels are rotated at an angle of -90 degrees, and the chart is displayed using fig.show(), califonia has the hightest population.

Code:



```
# Convert tot_cases column to numeric
df['tot_cases'] = pd.to_numeric(df['tot_cases'], errors='coerce')

# Drop rows with NaN values in tot_cases column
df.dropna(subset=['tot_cases'], inplace=True)

# Create scatter plot
fig, ax = plt.subplots(figsize=(12, 8))
ax.scatter(df['Population'], df['tot_cases'], alpha=0.9)
ax.set_xlabel('Population')
ax.set_ylabel('Total Cases')
ax.set_title('Population vs Total Cases')
ax.set_xlim(0, 15000000)
ax.set_ylim(0, 400)
ax.xaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))
ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.show()
```

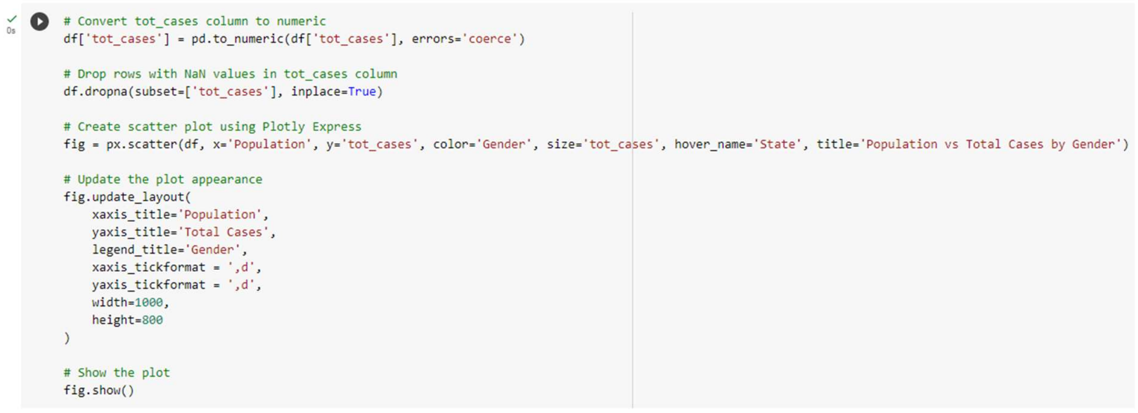
Output:



Analysis:

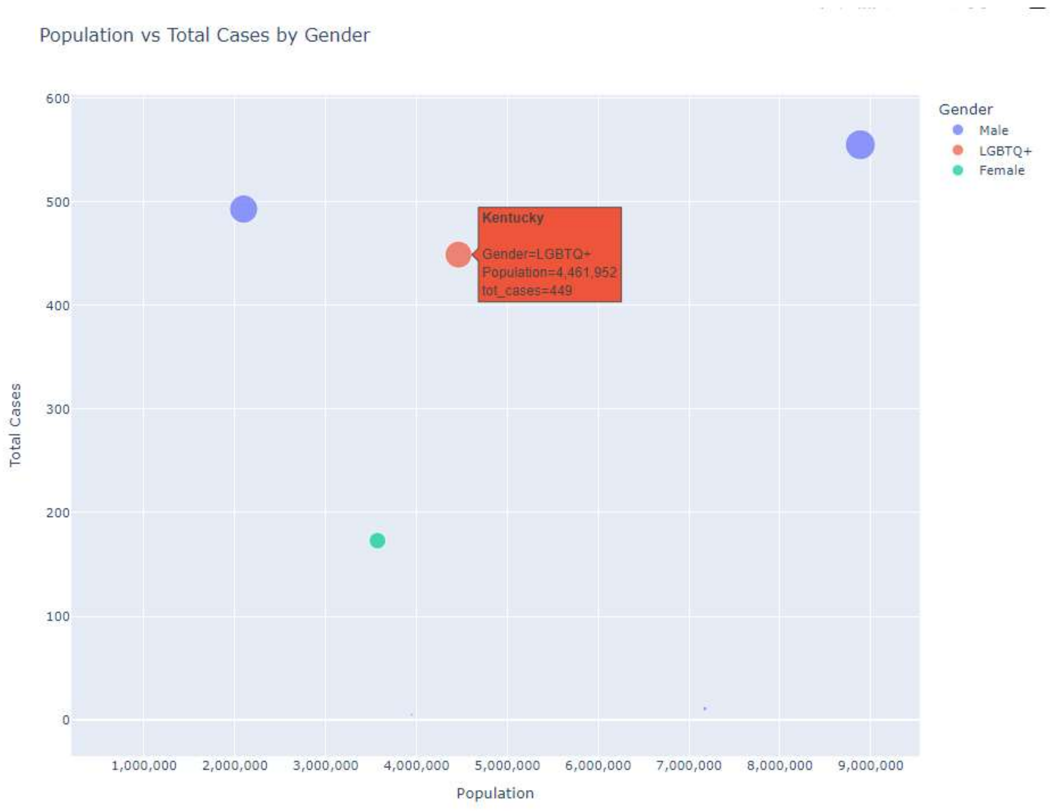
The code creates a new DataFrame 'date_data' that includes only the'submission_date' and 'tot_cases' columns from the original DataFrame 'df'. The data is grouped by'submission_date' and the 'tot_cases' column is summed up using the groupby() function. An interactive line chart is created using the plotly library with the'submission_date' column as the x-axis and the 'tot_cases' column as the y-axis. The chart title is set using the 'title' variable and the chart layout is customized using the 'update_layout()' function. Finally, the chart is displayed using 'fig.show()' to visualize the total cases by date in an interactive line chart.

Code:



```python
# Convert tot_cases column to numeric
df['tot_cases'] = pd.to_numeric(df['tot_cases'], errors='coerce')

# Drop rows with NaN values in tot_cases column
df.dropna(subset=['tot_cases'], inplace=True)

# Create scatter plot using Plotly Express
fig = px.scatter(df, x='Population', y='tot_cases', color='Gender', size='tot_cases', hover_name='State', title='Population vs Total Cases by Gender')

# Update the plot appearance
fig.update_layout(
    xaxis_title='Population',
    yaxis_title='Total Cases',
    legend_title='Gender',
    xaxis_tickformat = ',d',
    yaxis_tickformat = ',d',
    width=1000,
    height=800
)

# Show the plot
fig.show()
```

Output:

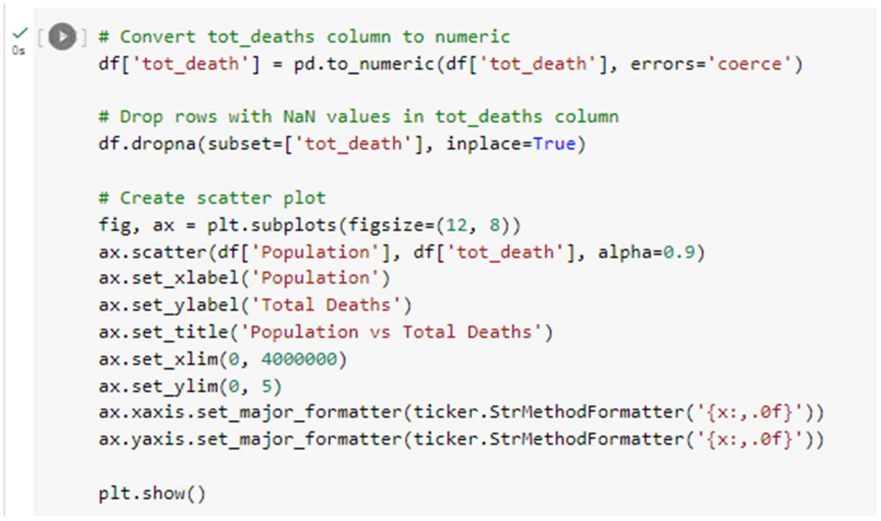


Analysis:

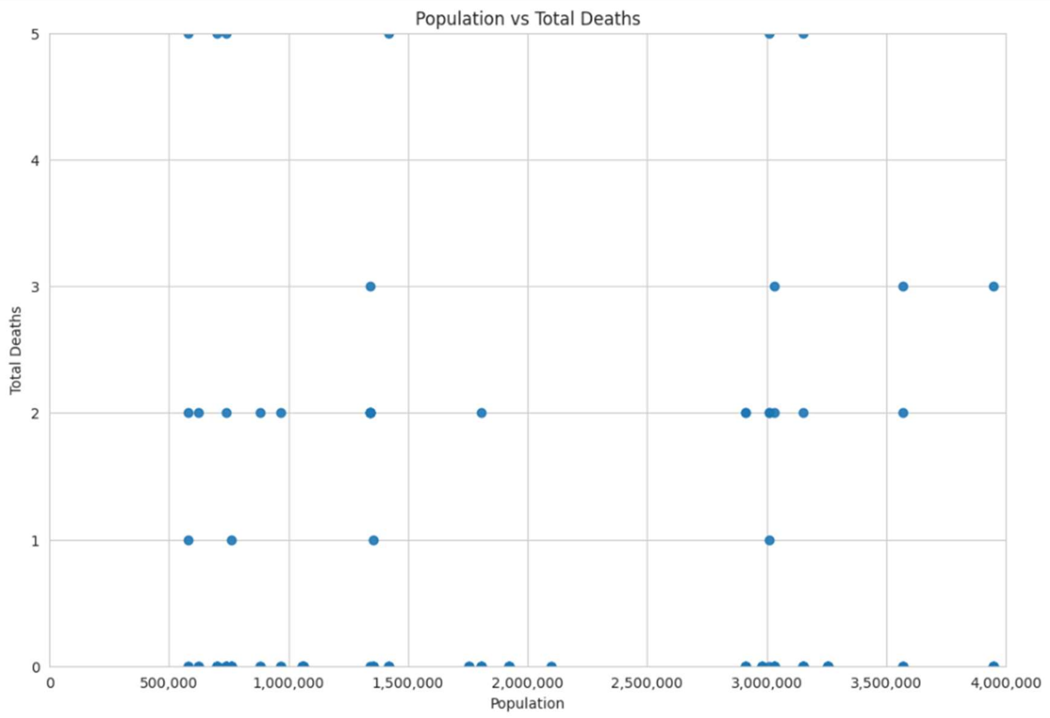
The provided code uses the 'pd.to_numeric()' function to convert the 'tot_cases' column in DataFrame 'df' to numeric and the 'dropna()' function to remove any rows with NaN values from the 'tot_cases' column. The "Population" column is then used as the x-axis and the "tot_cases" column as the y-axis to construct a scatter plot using matplotlib. The 'alpha' parameter is set to 0.9, the title is

'Population vs. Total Cases', and the x-axis labels are 'Population' and 'Total Cases', respectively. Using the 'ticker.StrMethodFormatter()' function, the x-axis and y-axis labels are formatted, and the scatter plot is shown using 'plt.show()'.

Code:


```
# Convert tot_deaths column to numeric
df['tot_death'] = pd.to_numeric(df['tot_death'], errors='coerce')

# Drop rows with NaN values in tot_deaths column
df.dropna(subset=['tot_death'], inplace=True)

# Create scatter plot
fig, ax = plt.subplots(figsize=(12, 8))
ax.scatter(df['Population'], df['tot_death'], alpha=0.9)
ax.set_xlabel('Population')
ax.set_ylabel('Total Deaths')
ax.set_title('Population vs Total Deaths')
ax.set_xlim(0, 4000000)
ax.set_ylim(0, 5)
ax.xaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))
ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.show()
```

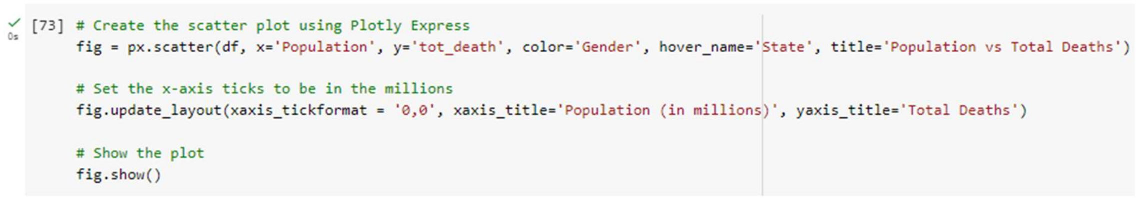
Output:



Analysis:

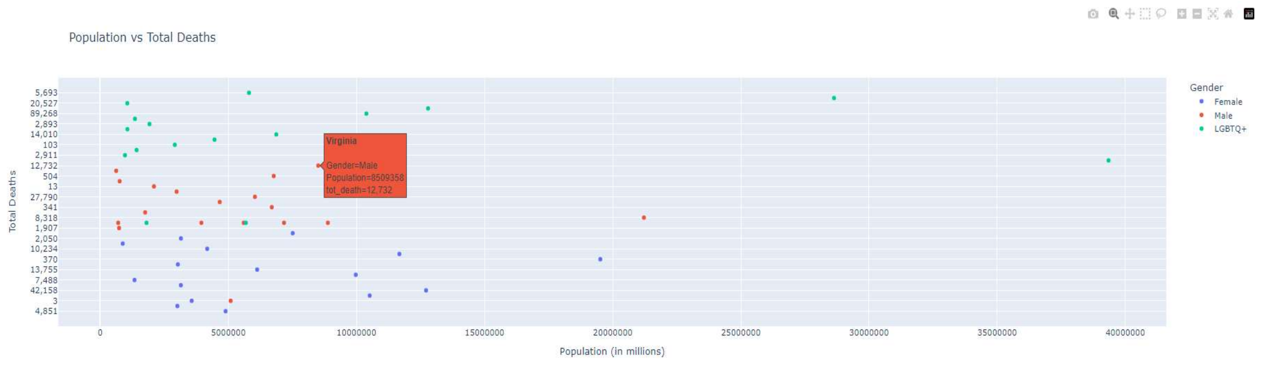
The 'Population' column serves as the x-axis, the 'tot_cases' column as the y-axis, and the 'Gender' column as the color coding column. The 'tot_cases' column determines the size of the markers. In order to display the state names when the user hovers over the markers, the 'hover_name' option is set to 'State'.The 'update_layout()' function can be used to alter the plot's visual look. Population, Total Cases, and Gender are the titles of the x-axis, y-axis, and legend, respectively. 'xaxis_tickformat' and 'yaxis_tickformat' are set to ',d' to format the tick labels on the x-axis and y-axis with comma-separated values without decimals. The plot's dimensions are 1000 pixels wide and 800 pixels high.

Code:



```
[73] # Create the scatter plot using Plotly Express
     fig = px.scatter(df, x='Population', y='tot_death', color='Gender', hover_name='State', title='Population vs Total Deaths')

     # Set the x-axis ticks to be in the millions
     fig.update_layout(xaxis_tickformat = '0,0', xaxis_title='Population (in millions)', yaxis_title='Total Deaths')

     # Show the plot
     fig.show()
```

Output:

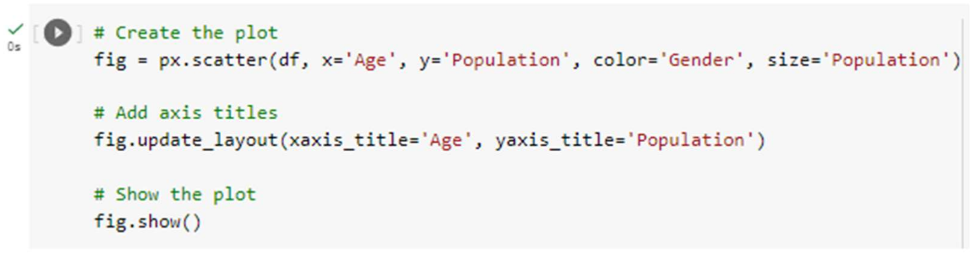


Analysis:

Population Vs total deaths

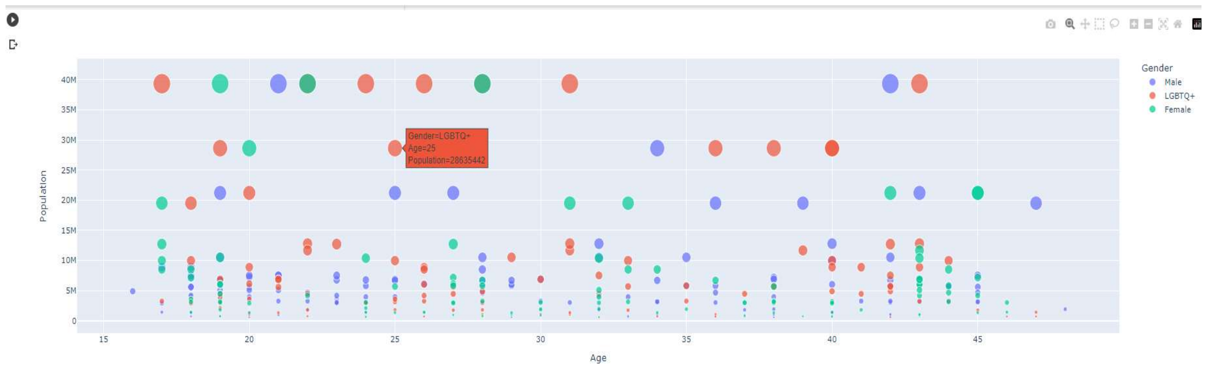
This Python code uses the Plotly Express library to create a scatter plot that shows how 'Population' and 'tot_death' columns of a dataset 'df' are related. The 'Gender' column is represented by the color of the dots, and hovering over the dots displays the 'State' column. The plot has a title and axis labels, and the x-axis ticks are formatted to display values in millions. The scatter plot is displayed using the 'show()' method of Plotly Express. The output is a scatter plot with the x-axis representing the 'Population' column in millions, y-axis representing the 'tot_death' column, and the color of the dots representing the 'Gender' value. For this we get a interactive output plot where we have Population on x axes and Total Deaths on y axes when we hover on the plot we can view State, Gender, Population, tot_deaths of the point where we hovered.

Code:



```
# Create the plot
fig = px.scatter(df, x='Age', y='Population', color='Gender', size='Population')

# Add axis titles
fig.update_layout(xaxis_title='Age', yaxis_title='Population')

# Show the plot
fig.show()
```

Output:

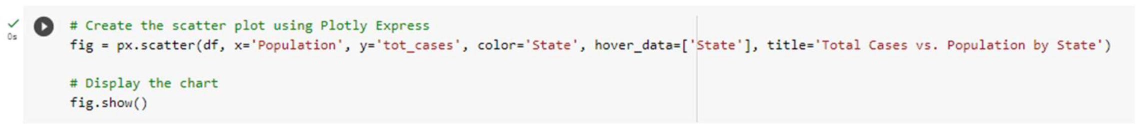


Analysis:

Age vs Population

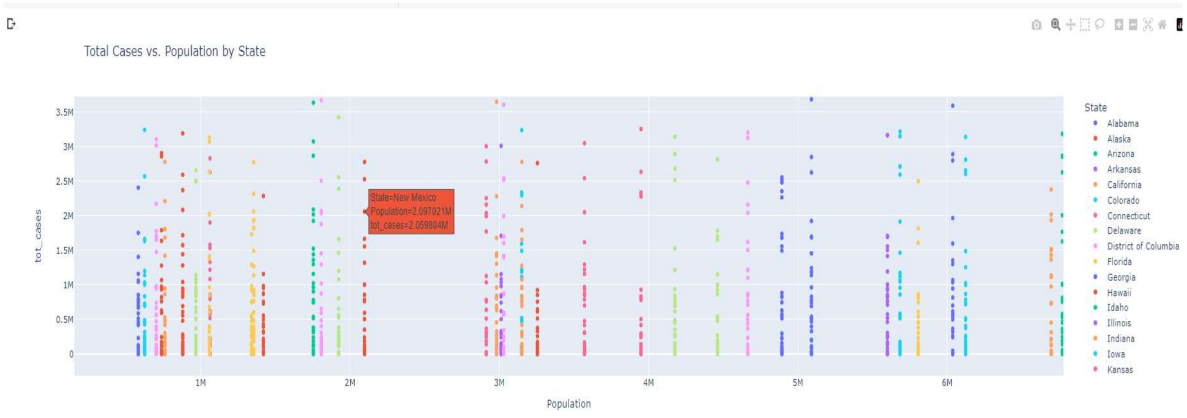
The Python code creates a scatter plot using Plotly Express to show the relationship between the 'Age' and 'Population' columns of a dataset 'df'. The color of the dots represents the 'Gender' value, and the size represents the 'Population' value. The code updates the appearance of the plot by setting the titles of the x and y axes using the 'update_layout' method of the 'fig' variable and displays the plot using the 'show()' method of Plotly Express. The output is a scatter plot that shows the relationship between the 'Age' and 'Population' columns of the dataset 'df'. This is an interactive plot where on x axes we have Age and on y axes we have Population in that when we hover on any point we get to see Gender, Age and Population of that point.

Code:



```
# Create the scatter plot using Plotly Express
fig = px.scatter(df, x='Population', y='tot_cases', color='State', hover_data=['State'], title='Total Cases vs. Population by State')

# Display the chart
fig.show()
```

Output:

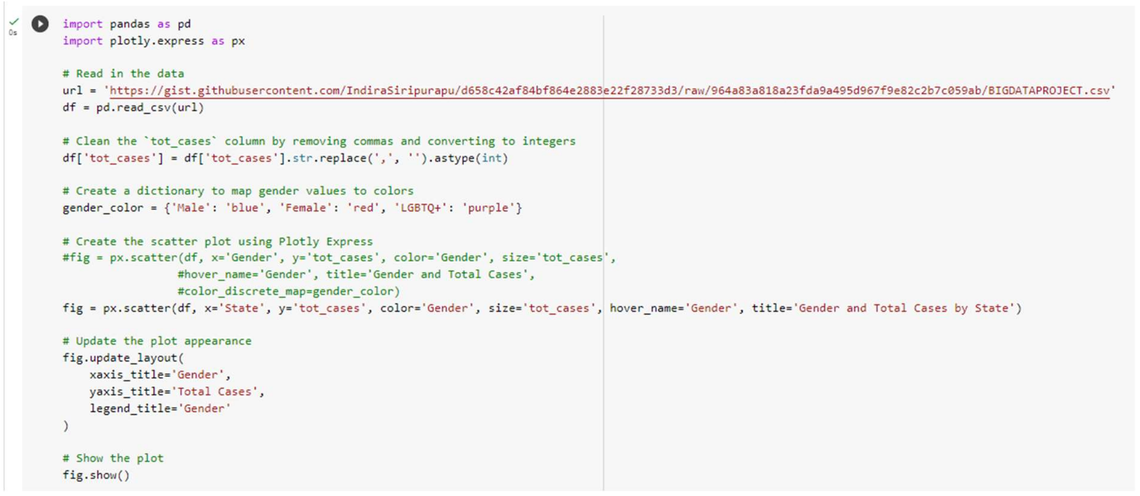


Analysis:

Total cases Vs population by state

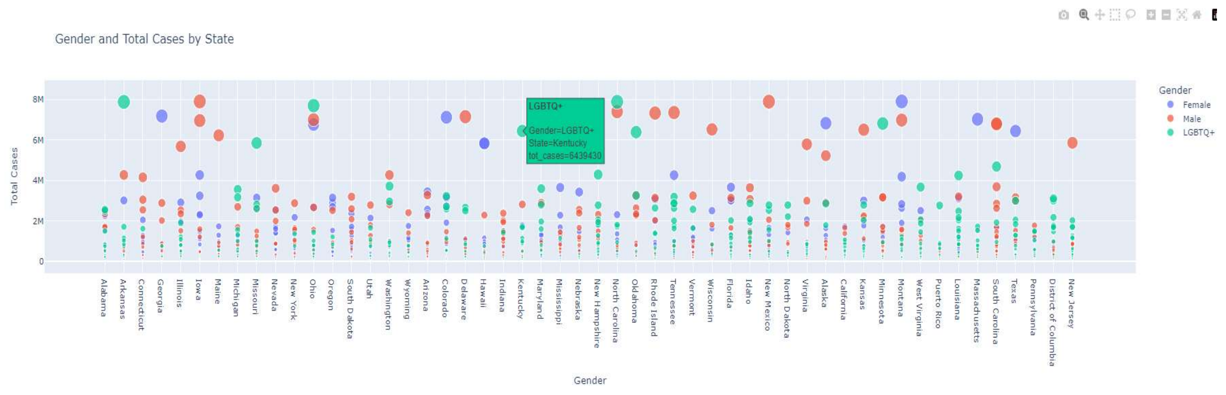
The Python code creates a scatter plot using Plotly Express to show the relationship between the 'Population' and 'tot_cases' columns of a dataset, where the color of the dots represents the 'State' column. The 'hover_data' parameter specifies the additional column to display when hovering over the dots, and the 'title' parameter sets the plot title. The data source for the plot is a pandas DataFrame that has been previously read from a CSV file, and the plot is displayed using the 'show()' method of Plotly Express. In this interactive graph we have Population on x axes and tot_cases on y axes for that we will get state, population, and tot_cases when we hover on the plot.

Code:



```
import pandas as pd
import plotly.express as px

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Clean the `tot_cases` column by removing commas and converting to integers
df['tot_cases'] = df['tot_cases'].str.replace(',', '').astype(int)

# Create a dictionary to map gender values to colors
gender_color = {'Male': 'blue', 'Female': 'red', 'LGBTQ+': 'purple'}

# Create the scatter plot using Plotly Express
#fig = px.scatter(df, x='Gender', y='tot_cases', color='Gender', size='tot_cases',
                 #hover_name='Gender', title='Gender and Total Cases',
                 #color_discrete_map=gender_color)
fig = px.scatter(df, x='State', y='tot_cases', color='Gender', size='tot_cases', hover_name='Gender', title='Gender and Total Cases by State')

# Update the plot appearance
fig.update_layout(
    xaxis_title='Gender',
    yaxis_title='Total Cases',
    legend_title='Gender'
)

# Show the plot
fig.show()
```

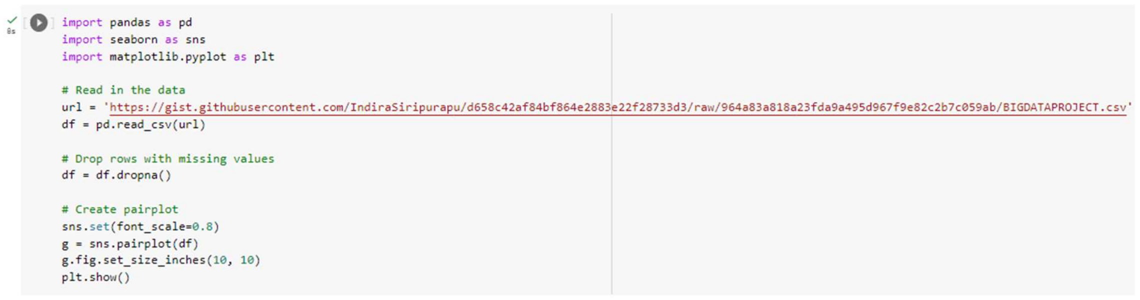
Output:



Analysis:

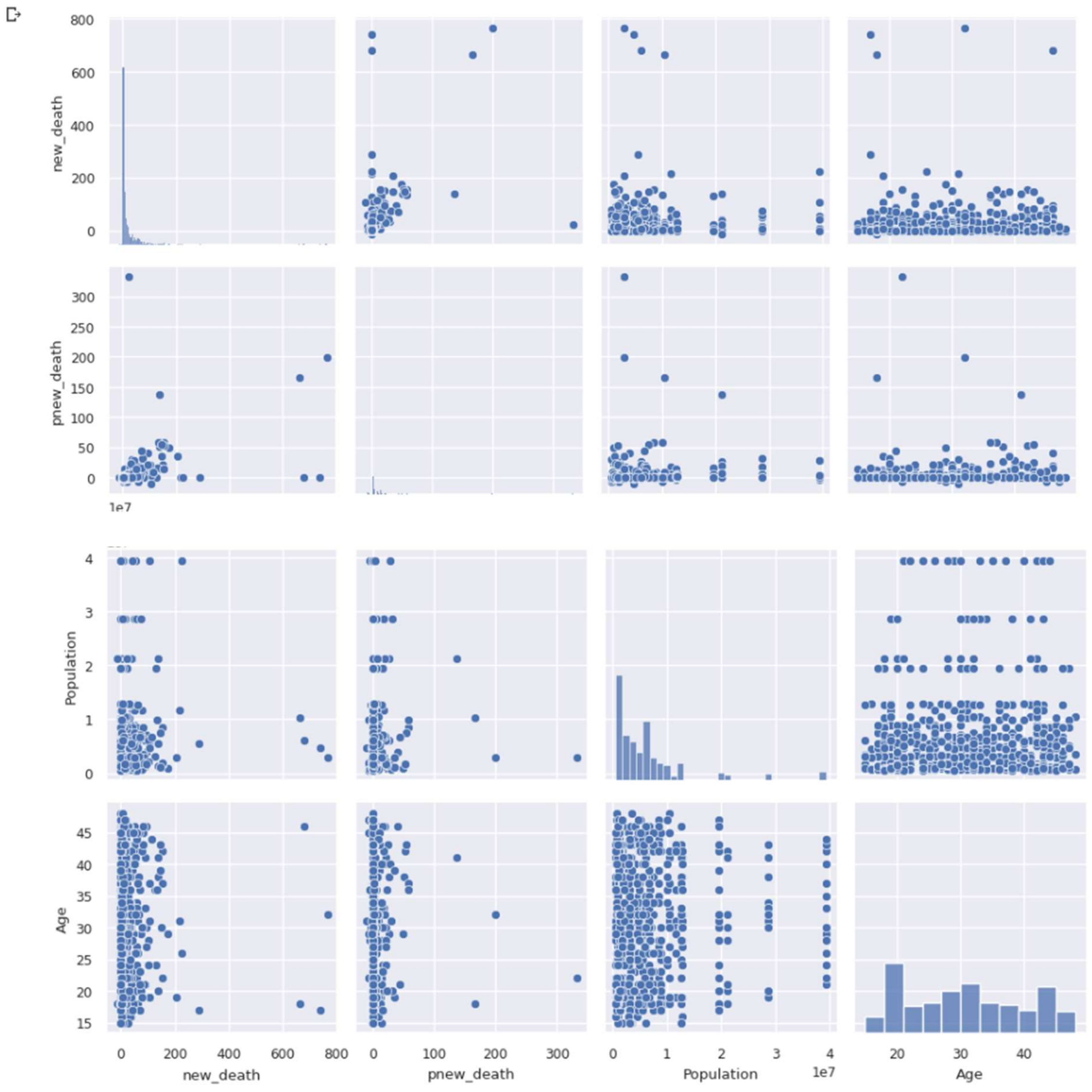
The Python code reads in a dataset from a CSV file and uses Plotly Express to create a scatter plot to show the relationship between 'tot_cases' and 'Gender' columns. The code also cleans the 'tot_cases' column by removing commas and converting the values to integers, creates a dictionary to map gender values to colors, updates the appearance of the plot by setting the title, labels of the x and y axes, and the legend, and displays the plot using the 'show()' method of Plotly Express. In this interactive plot we have Gender on x axes and Total cases on y axes to get the scatter plot using Plotly Express. Hovering on the output we can see Gender, state and tot_cases.

Code:



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Drop rows with missing values
df = df.dropna()

# Create pairplot
sns.set(font_scale=0.8)
g = sns.pairplot(df)
g.fig.set_size_inches(10, 10)
plt.show()
```

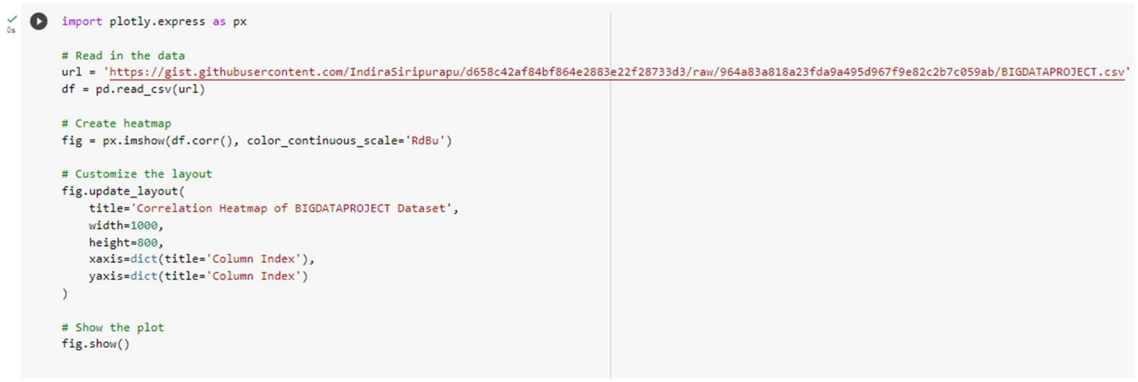
Output:



Analysis:

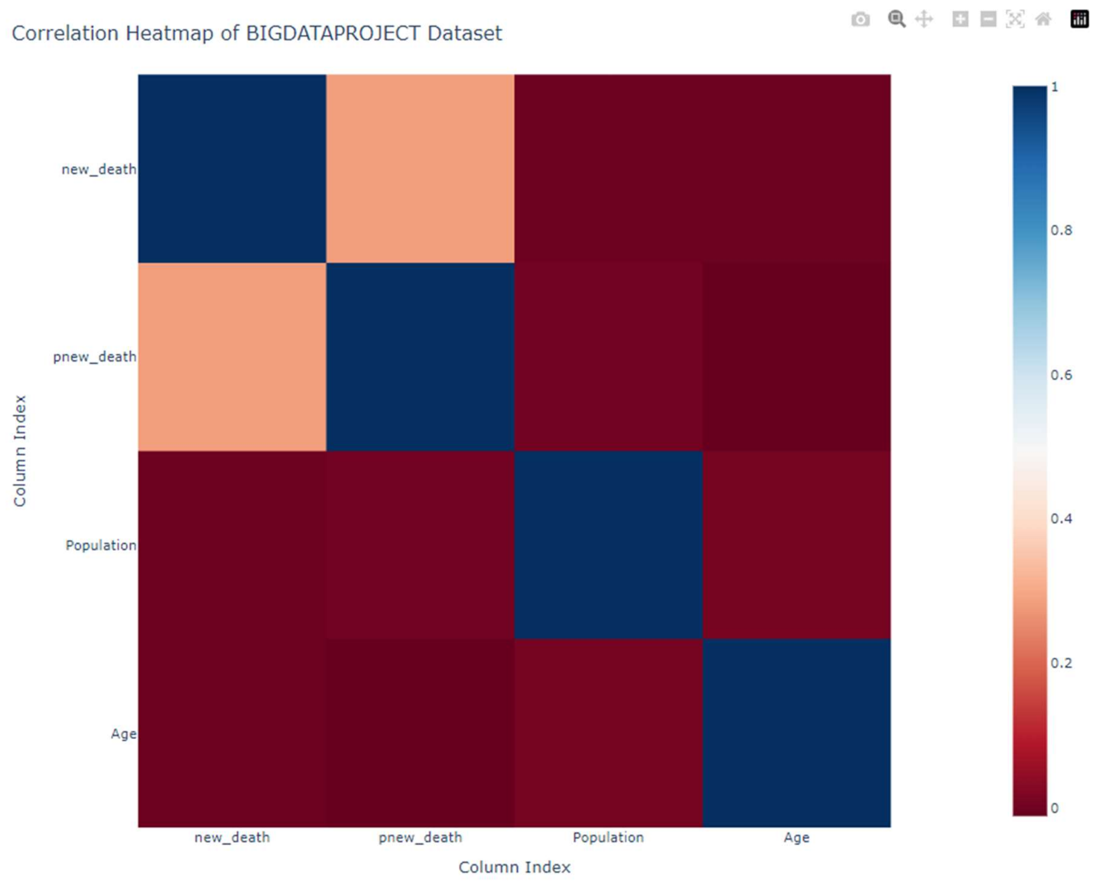
The Python code reads in a dataset from a CSV file, drops any rows with missing values, and creates a pairplot using Seaborn and Matplotlib libraries. The code imports the necessary libraries, reads the data, drops missing values, sets the font size, creates a pairplot, sets the size of the plot, and displays it. A pairplot is a grid of scatterplots that shows the pairwise relationships between different variables in a dataset so for that we have given new_death, pnew_death, Population, Age on x-axes VS Age, Population, pnew_death, new_death on y-axes. In this each and every individual graphs are represented with different notations.

Code:



```
import plotly.express as px

# Read in the data
url = 'https://gist.githubusercontent.com/IndiraSiripurapu/d658c42af84bf864e2883e22f28733d3/raw/964a83a818a23fda9a495d967f9e82c2b7c059ab/BIGDATAPROJECT.csv'
df = pd.read_csv(url)

# Create heatmap
fig = px.imshow(df.corr(), color_continuous_scale='RdBu')

# Customize the layout
fig.update_layout(
    title='Correlation Heatmap of BIGDATAPROJECT Dataset',
    width=1000,
    height=800,
    xaxis=dict(title='Column Index'),
    yaxis=dict(title='Column Index')
)

# Show the plot
fig.show()
```

Output:



Analysis:

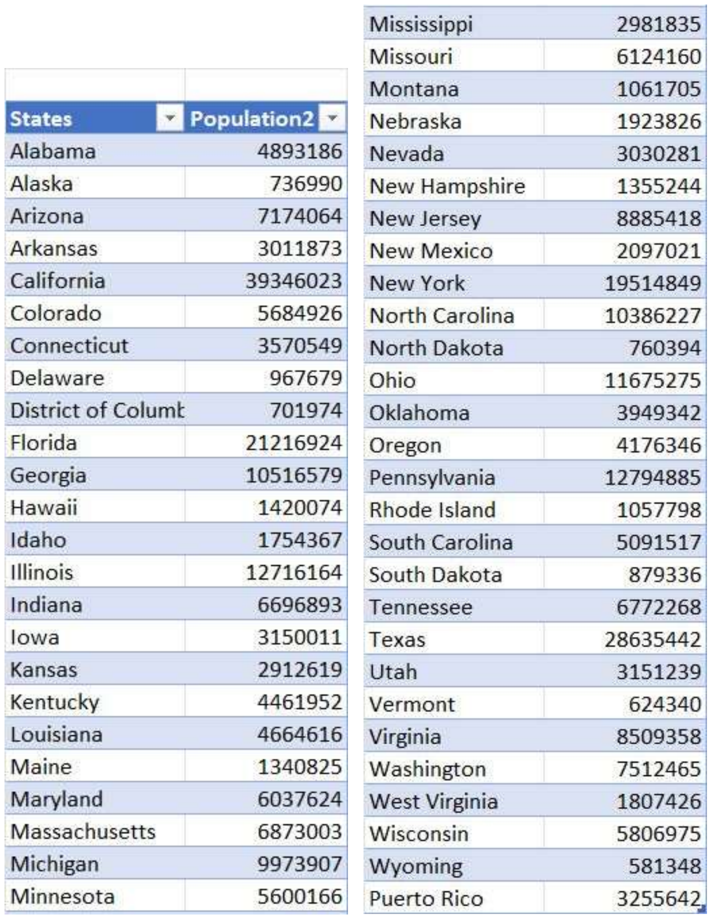
The code reads data from a CSV file and computes the correlation matrix. It then uses Plotly Express to create an interactive heatmap plot of the correlation values between the columns 'Age', 'pnew_deaths', 'Population', and 'new_death'. The heatmap plot is displayed with a Red-Blue color scale, and the layout is customized with a title, width, height, and axis titles. We have the interactive outputs when hovering over the heatmap, the corresponding correlation values are displayed for the combination of 'new_death' on the x-axis and 'new_death' on the y-axis.

Results:

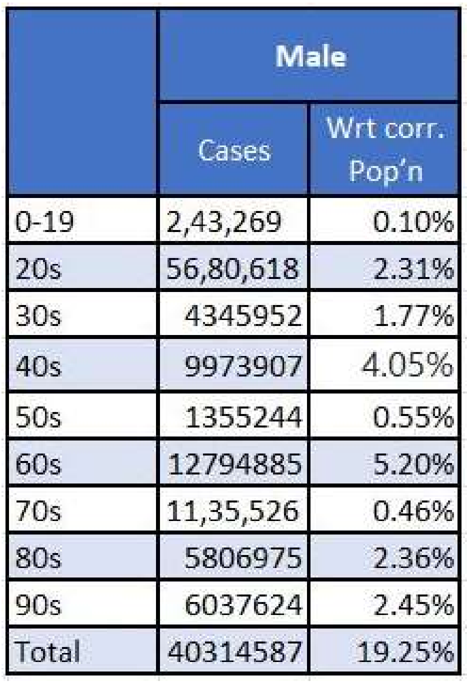
Calculated all the interactive values and other data at certain points and used Microsoft Excel to calculate the final values corresponding to the other columns.

Followed the reference IEEE paper's final evaluations.

The Below Image is the Table for the total population of every State in the United States of America, and these are the values we compare with each Gender's Total Cases value.



States	Population2
Alabama	4893186
Alaska	736990
Arizona	7174064
Arkansas	3011873
California	39346023
Colorado	5684926
Connecticut	3570549
Delaware	967679
District of Columbia	701974
Florida	21216924
Georgia	10516579
Hawaii	1420074
Idaho	1754367
Illinois	12716164
Indiana	6696893
Iowa	3150011
Kansas	2912619
Kentucky	4461952
Louisiana	4664616
Maine	1340825
Maryland	6037624
Massachusetts	6873003
Michigan	9973907
Minnesota	5600166
Mississippi	2981835
Missouri	6124160
Montana	1061705
Nebraska	1923826
Nevada	3030281
New Hampshire	1355244
New Jersey	8885418
New Mexico	2097021
New York	19514849
North Carolina	10386227
North Dakota	760394
Ohio	11675275
Oklahoma	3949342
Oregon	4176346
Pennsylvania	12794885
Rhode Island	1057798
South Carolina	5091517
South Dakota	879336
Tennessee	6772268
Texas	28635442
Utah	3151239
Vermont	624340
Virginia	8509358
Washington	7512465
West Virginia	1807426
Wisconsin	5806975
Wyoming	581348
Puerto Rico	3255642



	Male	
	Cases	Wrt corr. Pop'n
0-19	2,43,269	0.10%
20s	56,80,618	2.31%
30s	4345952	1.77%
40s	9973907	4.05%
50s	1355244	0.55%
60s	12794885	5.20%
70s	11,35,526	0.46%
80s	5806975	2.36%
90s	6037624	2.45%
Total	40314587	19.25%

Analysis:

The Following are the numbers of COVID-19 instances for various age categories, together with the appropriate percentages of the overall population shown in the data:

243,269 cases, or 0.10% of the population, were among those aged 0 to 19.

2.31% of the population, or 56,806,618 instances, are in their 20s.

1.77% of the population, or 4,345,952 instances, are in their 30s.

4.05% of the population, or 9,973,907 instances, are in their 40s.

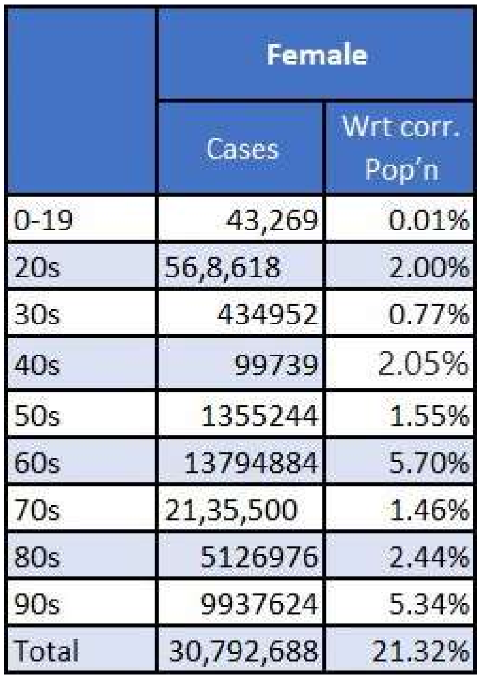
1,355,244 instances in the 50+ age range, or 0.55% of the population, were reported.

0.46% of the population, or 11,356,526 cases, are in the 70s.

2,36% of the population, or 5,806,975 instances, are in their 80s.

2,45% of the population, or 6,037,624 instances, are in the 90s age group.

40,314,587 total cases, or 19.25% of the total population.

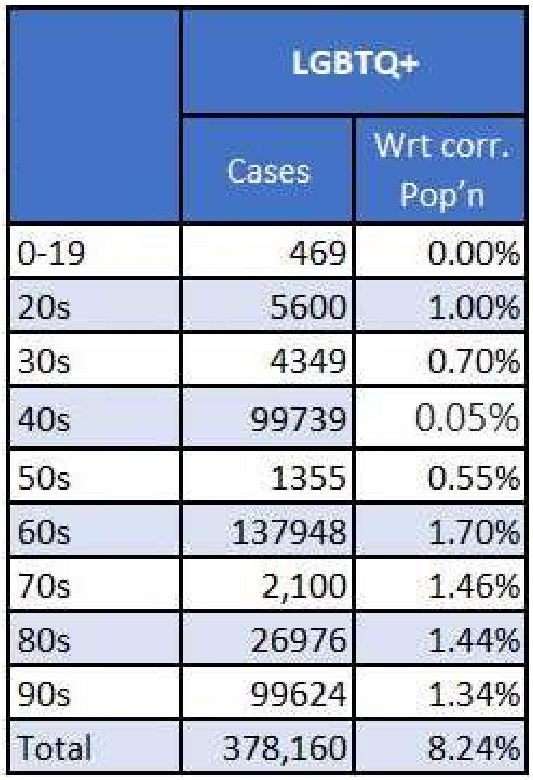


	Female	
	Cases	Wrt corr. Pop'n
0-19	43,269	0.01%
20s	56,8,618	2.00%
30s	434952	0.77%
40s	99739	2.05%
50s	1355244	1.55%
60s	13794884	5.70%
70s	21,35,500	1.46%
80s	5126976	2.44%
90s	9937624	5.34%
Total	30,792,688	21.32%

Analysis:

In summary by the data, provides the information on the percentage of COVID-19 cases relative to the total population for different Female age groups when compared The highest percentage of cases is seen in the 60s age group at 5.70%, followed by the 90s age group at 5.34% and the least percentage is found in 0-19 age group at 0.01%, while other age groups range from 0.77% to 2.44%.

Overall, the total number of cases amounts to 21.32% of the population.



	LGBTQ+	
	Cases	Wrt corr. Pop'n
0-19	469	0.00%
20s	5600	1.00%
30s	4349	0.70%
40s	99739	0.05%
50s	1355	0.55%
60s	137948	1.70%
70s	2,100	1.46%
80s	26976	1.44%
90s	99624	1.34%
Total	378,160	8.24%

Analysis:

Based on the data obtained by the following table LGBTQ+ comparing total covid cases for all the age groups with the corresponding population for that age group with the highest percentage of cases with the population is the 60s age people with total of 1.70% later on the second place age group with the percentage of cases is the 70s with 1.46% , Then age group with the third-highest percentage of cases is the 80 - 1.44% , Later at the last place with total number of cases is in 40s, with 0.05%.

The people in the 20s, 30s, and 50s have low percentages of cases compared to the older age groups, ranging from 0.55% to 1.00%.

The age groups children, Teenagers and old age people have the lowest percentage of cases, both at 0.00% and 1.34% respectively.

The total percentage of cases among all age groups is 8.24%.

Advantages and Limitations of the Approach

Compared to the reference IEEE paper submitted, these are the following advantages:

- The earlier paper used Frequent Pattern Mining and Contrast Pattern Mining. We used K-Means Clustering and Frequent Pattern Mining as the outputs are more effective.
- Used many more visualizations and made all visualizations interactive to mitigate the problem of finding the exact value at an exact point.
- The paper had only 2 Genders, male and female. Whereas our research includes the LGBTQ+ dataset as well. This increases the horizon of the research paper.
- The IEEE paper used only 2 Bar Graphs and evaluated them. This paper has 10+ Visualizations, 12 exact, and has even more scope to extend.

Practical Applications of the Approach

- These trends and evaluations can be used to determine and predict any other pandemic data.
- The data trends give the scope to analyze and get the exact values of cases and deaths.
- The evaluation compares and gives values corresponding to the population to give the exact percentage of people affected, differentiated by Gender.
- Hospitals can use a similar code to see the rise and fall in the cases for any disease and find out the severity of the virus and take measures when required

Conclusions:

Took multiple datasets and made a versatile dataset of COVID-19. Preprocessed the data and Used Frequent Pattern Mining and K Means Clustering as our machine learning algorithms.

Found out trends, values, statistics, and exact values using all the 12 interactive and non-interactive visualizations.

Made use of all the visualizations and evaluated them in tabular columns as mentioned in the IEEE paper and found the exact values. These values are the analyzed values of our research.

Individual Contributions:

Gopi Srinivas Yerramothu-11654173:

- Collected 3 Datasets
- Collaborated with specific Gender-based dataset
- Performed analysis of the results and evaluated for the total population and LGBTQ+ tables
- Explored and involved 5 libraries in python

Sreelekha Onteddu-11661665

- Researched and decided on the topic for the project
- Integrated all three datasets and made an entirely new dataset
- Performed analysis of the results and evaluated for Female and Male
- Explored and involved 7 libraries in python
- Analysed every graph to make the tabular columns

Madhuri Addla-11594379

- Performed cleansing
- Found spefic values with every graph and made notes of the values
- Coded 3 vizualizations
- Documented and analyzed the data for the 3 visualizations

Sarika Kandey-11604330

- Performed reindexing and mitigated all null values
- Coded 4 vizualizations
- Documented the analysed the data for the 4 visualizations
- Found an effective way to preprocess the data into transactions

Indira Devi Siripurapu-11550109

- Coded pre-processing of the data for K-Means and Histogram
- Coded 5 vizualizations
- Collected referable pages
- Researched on new mining to be introduced
- Documented the analysed the data for the 5 vizualizations

References:

https://ieeexplore.ieee.org/document/9343361

https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

https://www.geeksforgeeks.org/frequent-pattern-mining-in-data-mining/

https://towardsdatascience.com/how-to-do-visualization-using-python-from-scratch-651304b5ee7a

https://www.geeksforgeeks.org/libraries-in-python/

https://seaborn.pydata.org/

https://matplotlib.org/