```python
import pandas as pd


# Load the dataset into a DataFrame
covid_data = pd.read_csv('usacoviddata.csv')  # Replace
'your_file_path.csv' with the actual file path

# Display the first few rows of the dataset
print(covid_data.head())
```

```
    USA State  Total Cases  Total Deaths  Total Recovered  Active
Cases  \
0     Alabama      1659936         21138        1623935.0
14863.0
1      Alaska       301513          1485         298902.0
1126.0
2     Arizona      2508168         33749        2461333.0
13086.0
3    Arkansas      1029976         13246        1012777.0
3953.0
4  California     12393361        104634       12227662.0
61065.0

   Tot Cases/ 1M pop  Deaths/ 1M pop  Total Tests  Tests/ 1M pop
Population
0              338542            4311      9332317        1903317
4903185
1              412159            2030      4790640        6548661
731545
2              344589            4637     22489035        3089698
7278717
3              341300            4389      8036831        2663139
3017804
4              313659            2648    202799362        5132573
39512223
```

```python
print(covid_data.isnull().sum())
```

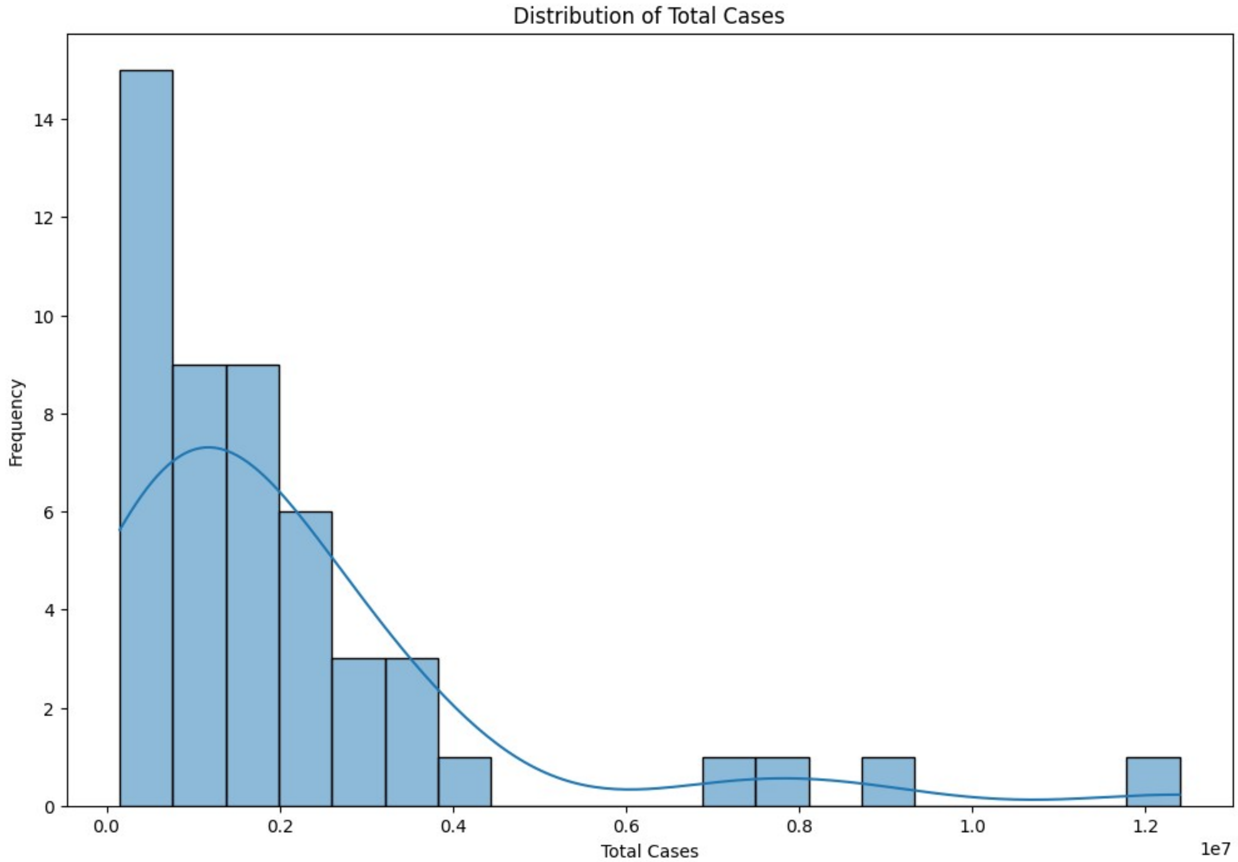
```
USA State            0
Total Cases          0
Total Deaths         0
Total Recovered      5
Active Cases         5
Tot Cases/ 1M pop    0
Deaths/ 1M pop       0
Total Tests          0
Tests/ 1M pop        0
Population           0
dtype: int64
```

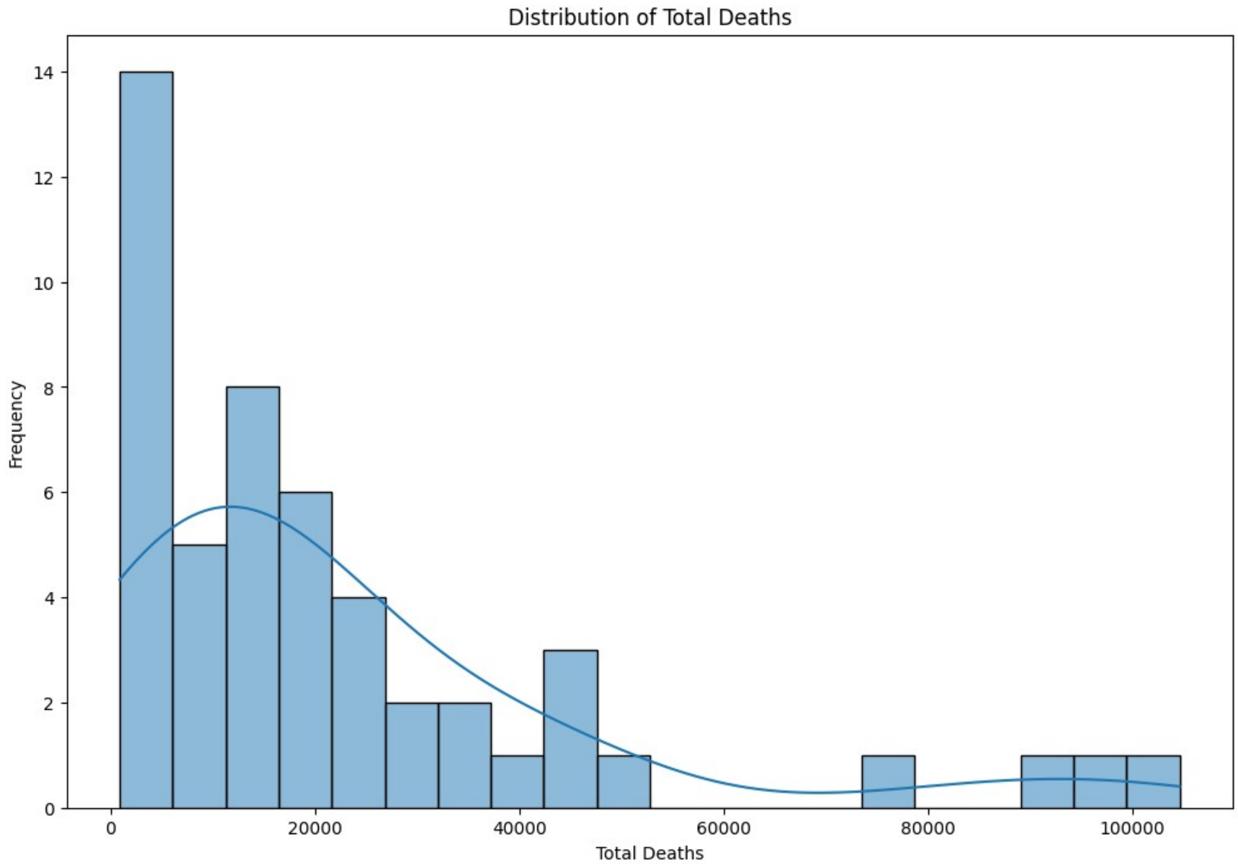
```python
# Fill missing values with zeros
covid_data['Total Recovered'].fillna(0, inplace=True)
covid_data['Active Cases'].fillna(0, inplace=True)

# Check if missing values are handled
print(covid_data.isnull().sum())
```

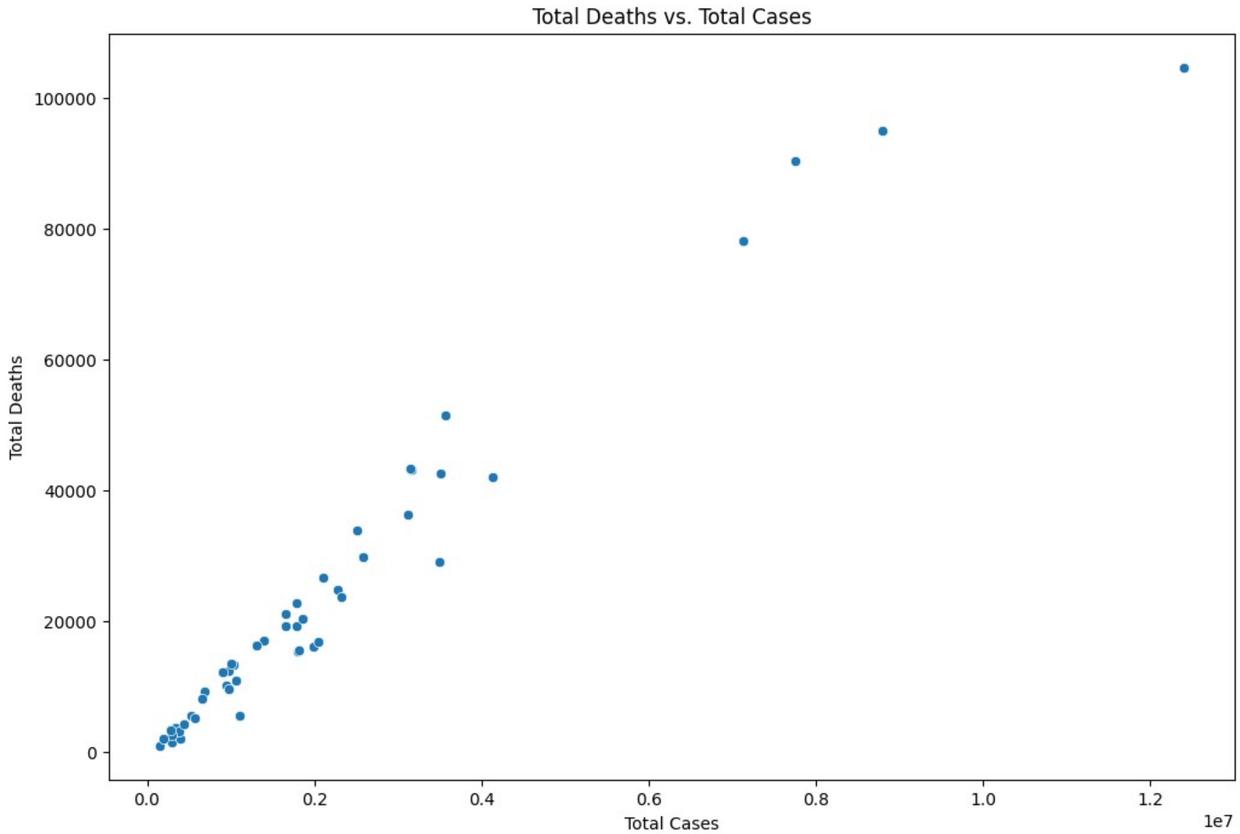
```
USA State            0
Total Cases          0
Total Deaths         0
Total Recovered      0
Active Cases         0
Tot Cases/ 1M pop    0
Deaths/ 1M pop       0
Total Tests          0
Tests/ 1M pop        0
Population           0
dtype: int64
```

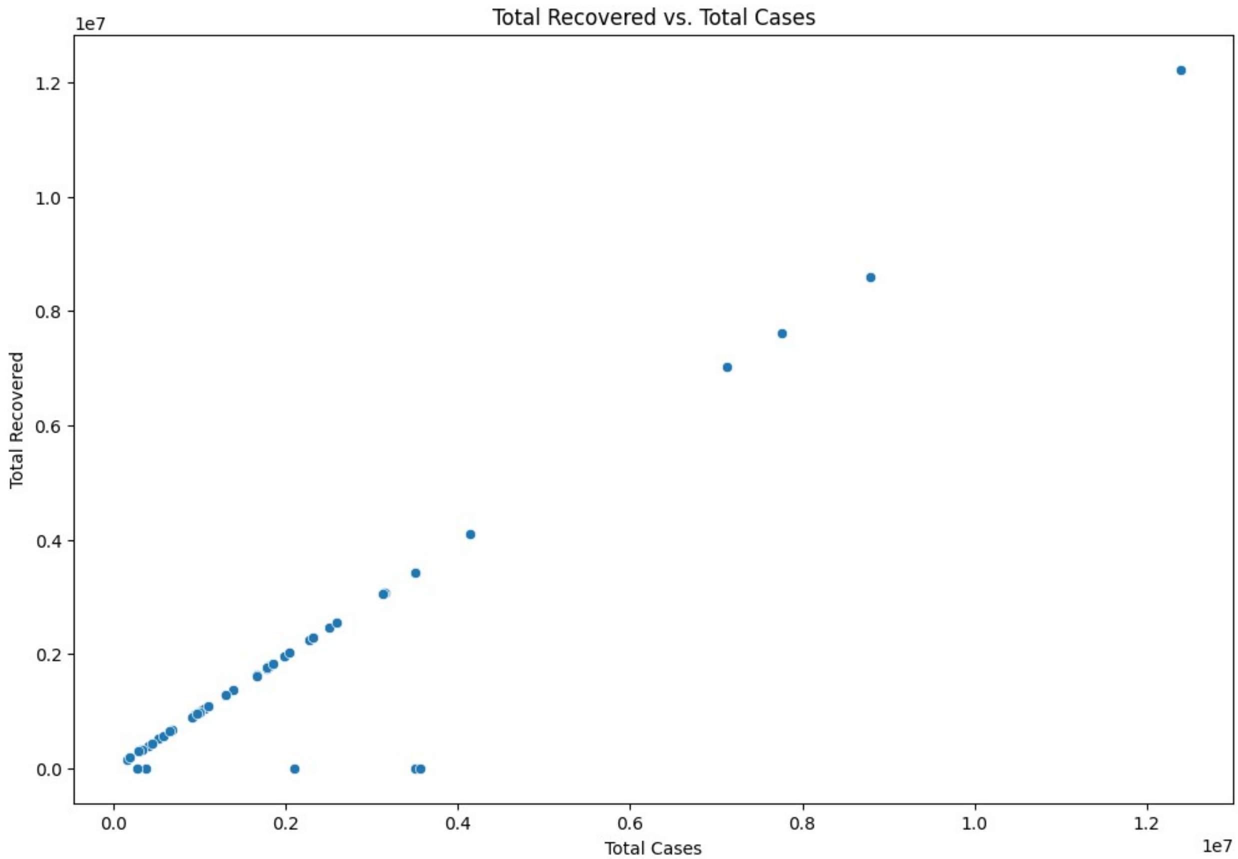
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 8))
sns.histplot(covid_data['Total Cases'], bins=20, kde=True)
plt.title('Distribution of Total Cases')
plt.xlabel('Total Cases')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(12, 8))
sns.histplot(covid_data['Total Deaths'], bins=20, kde=True)
plt.title('Distribution of Total Deaths')
plt.xlabel('Total Deaths')
plt.ylabel('Frequency')
plt.show()

# Explore relationships between different features (e.g., Total Cases
vs. Total Deaths)
plt.figure(figsize=(12, 8))
sns.scatterplot(x='Total Cases', y='Total Deaths', data=covid_data)
plt.title('Total Deaths vs. Total Cases')
plt.xlabel('Total Cases')
plt.ylabel('Total Deaths')
plt.show()

plt.figure(figsize=(12, 8))
sns.scatterplot(x='Total Cases', y='Total Recovered', data=covid_data)
plt.title('Total Recovered vs. Total Cases')
plt.xlabel('Total Cases')
```

```
plt.ylabel('Total Recovered')
plt.show()
```




Distribution of Total Deaths
Frequency
Total Deaths
0
2
4
6
8
10
12
14
0
20000
40000
60000
80000
100000


Total Deaths vs. Total Cases
Total Deaths
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Total Cases
1e7




```
import numpy as np
df = pd.DataFrame(covid_data)

# 1. Create New Features
df['Case Fatality Rate'] = df['Total Deaths'] / df['Total Cases']
df['Recovery Rate'] = df['Total Recovered'] / df['Total Cases']
df['Active Cases Percentage'] = df['Active Cases'] / df['Total Cases']

# 2. Transform Variables
# Apply log transformation to skewed variables
skewed_vars = ['Total Cases', 'Total Deaths', 'Total Recovered',
'Active Cases', 'Total Tests']
df[skewed_vars] = np.log1p(df[skewed_vars])



# Display the updated DataFrame
print(df)

         USA State  Total Cases  Total Deaths  Total Recovered  Active
Cases  \
0          Alabama     2.729309      2.394150         2.727877
2.361486
1           Alaska     2.611288      2.116718         2.610649
```

```
2.082850
2          Arizona     2.755892      2.435956          2.754693
2.349409
3         Arkansas     2.697666      2.350568          2.696531
2.228129
4       California     2.852593      2.530376          2.851816
2.486548
5         Colorado     2.734644      2.364508          2.733729
2.321940
6      Connecticut     2.694562      2.343901          2.693575
2.145824
7         Delaware     2.619366      2.213773          2.618270
2.122860
8          Florida     2.825197      2.518515          2.824073
2.478359
9          Georgia     2.770417      2.457123          2.768763
2.448782
10          Hawaii     2.632165      2.152447          2.630937
2.248603
11           Idaho     2.651319      2.262743          2.650259
2.170928
12        Illinois     2.787194      2.454926          2.786565
0.000000
13         Indiana     2.744701      2.415146          0.000000
0.000000
14            Iowa     2.699490      2.330892          2.697781
2.366923
15          Kansas     2.691961      2.325626          2.691108
2.126027
16        Kentucky     2.734131      2.385015          2.732309
2.426460
17       Louisiana     2.729310      2.385159          2.727408
2.420921
18           Maine     2.617200      2.201958          2.616059
2.149218
19        Maryland     2.717860      2.373676          2.716756
2.275135
20   Massachusetts     2.749599      2.408833          2.748709
2.280999
21        Michigan     2.770125      2.457498          2.768915
2.373018
22       Minnesota     2.735015      2.364997          2.734347
2.202171
23     Mississippi     2.695703      2.352193          2.694589
2.193621
24        Missouri     2.733882      2.400945          2.733045
0.000000
25         Montana     2.618722      2.221331          2.617835
1.913173
```

26	Nebraska	2.657534	2.254435	2.656622	2.170831
27	Nevada	2.689283	2.342112	2.687875	2.280687
28	New Hampshire	2.628517	2.201566	0.000000	0.000000
29	New Jersey	2.769737	2.442231	2.768365	2.429471
30	New Mexico	2.669454	2.315597	2.667261	2.340915
31	New York	2.820116	2.506815	2.819201	2.426695
32	North Carolina	2.776872	2.422776	0.000000	0.000000
33	North Dakota	2.609325	2.178113	2.608561	1.980998
34	Ohio	2.776931	2.455976	2.775604	2.429519
35	Oklahoma	2.713552	2.369325	2.712640	2.130465
36	Oregon	2.693959	2.318830	2.692691	2.307986
37	Pennsylvania	2.778000	2.472241	0.000000	0.000000
38	Rhode Island	2.639281	2.233766	2.638533	1.964483
39	South Carolina	2.736633	2.390501	2.735550	2.327973
40	South Dakota	2.606596	2.206169	0.000000	0.000000
41	Tennessee	2.757884	2.424810	2.756913	2.319169
42	Texas	2.832589	2.522548	2.831275	2.526022
43	Utah	2.702481	2.261520	2.701869	2.245309
44	Vermont	2.560692	2.058624	2.560190	1.657941
45	Virginia	2.750807	2.404717	2.750081	2.173835
46	Washington	2.741058	2.369290	2.740095	2.351092
47	West Virginia	2.666992	2.303639	2.665572	2.255425
48	Wisconsin	2.742796	2.372735	2.742060	2.279330
49	Wyoming	2.576324	2.154677	2.575300	1.969851

```
    Tot Cases/ 1M pop  Deaths/ 1M pop  Total Tests  Tests/ 1M pop
Population  \
0              338542            4311     2.836091        1903317
4903185
1              412159            2030     2.796194        6548661
731545
2              344589            4637     2.886394        3089698
7278717
3              341300            4389     2.827287        2663139
3017804
4              313659            2648     3.002098        5132573
39512223
5              312870            2665     2.883613        3715530
5758736
6              275897            3465     2.869902        4704585
3565287
7              345794            3556     2.697406        1053651
973764
8              361221            4201     2.945345        3110202
21477737
9              297388            4059     2.900317        2723446
10617423
10             283819            1419     2.786663        2896566
1415872
11             294239            3068     2.773210        1847542
1787065
12             326446            3315     2.937796        4569040
12671821
13             312718            3962     2.883914        3195405
6732219
14             335420            3422     2.830545        2691723
3155070
15             324910            3511     2.811962        2133427
2913314
16             400111            4283     2.858256        3060940
4467673
17             357075            4122     2.876754        4073167
4648794
18             243167            2314     2.809398        4430910
1344212
19             230619            2800     2.892491        4150897
6045680
20             329696            3607     2.930438        7314755
6892503
21             314696            4334     2.900370        2898207
9986857
22             321306            2736     2.890960        4328671
5639632
23             336144            4527     2.819620        2373410
```

```
2976149
24          290140          3711     2.862129          2384091
6137428
25          312280          3473     2.762614          2609534
1068778
26          296938          2617     2.804654          2846355
1934408
27          295415          3937     2.818009          2232203
3080156
28          280952          2279     2.796884          3563370
1359711
29          351650          4083     2.906587          3649900
8882190
30          325027          4405     2.831355          4106238
2096829
31          366140          4015     2.980342          6759879
19453561
32          333846          2771     2.895831          2541556
10488084
33          385231          3298     2.754723          3231338
762062
34          299830          3638     2.891761          2118811
11689100
35          330139          4083     2.804296          1383274
3956971
36          231149          2263     2.853127          2965691
4217737
37          278511          4021     2.904071          2418574
12801989
38          418935          3933     2.832696          8314785
1059361
39          360838          3945     2.876884          3686182
5148714
40          319779          3652     2.751225          2634917
884659
41          378983          4354     2.862948          2173560
6829174
42          303225          3272     2.950645          2548789
28995881
43          345133          1690     2.836558          2934181
3205958
44          246814          1489     2.787900          6705820
623989
45          271311          2782     2.877455          2246206
8535519
46          261261          2121     2.876610          2480278
7614893
47          367023          4569     2.821509          4068442
1792147
```

```
48          351028          2878    2.878819      3373719
5822434
49          326303          3539    2.725750      2716174
578759
```

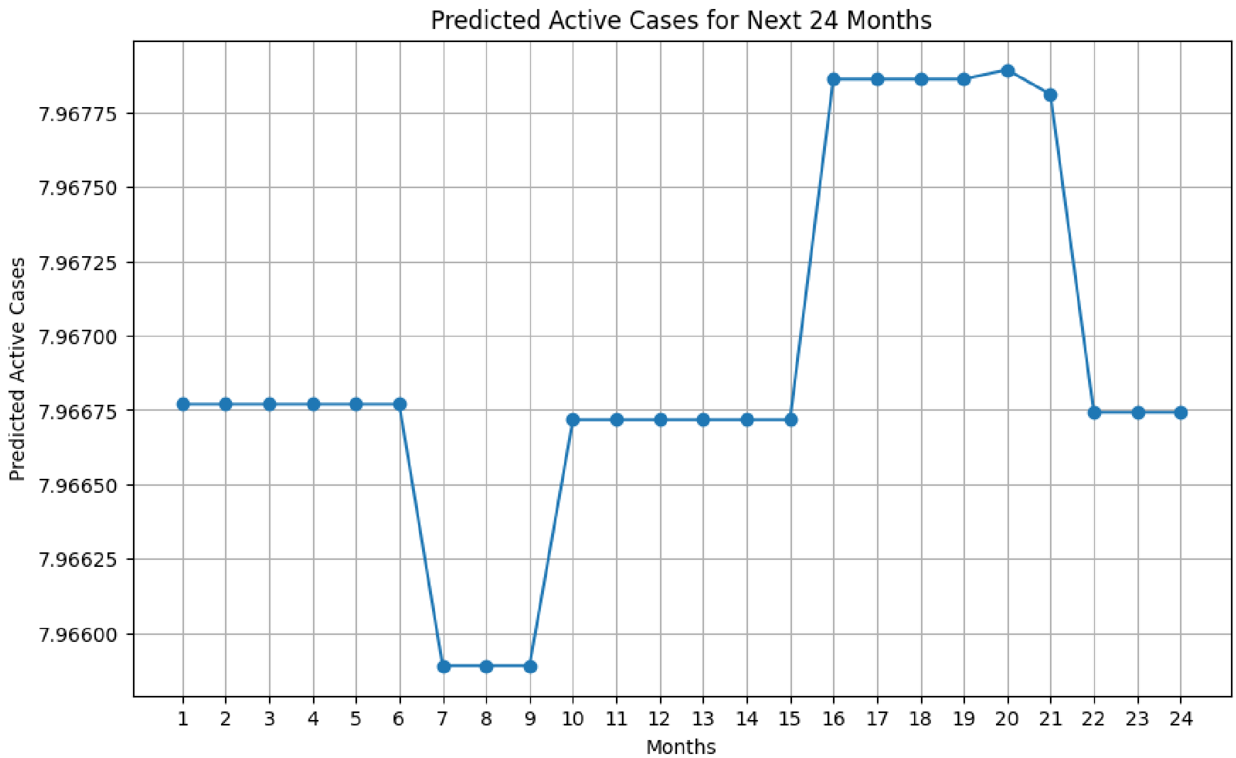
	Case Fatality Rate	Recovery Rate	Active Cases Percentage
0	0.695341	0.998469	0.670751
1	0.578909	0.999311	0.556991
2	0.707614	0.998721	0.643321
3	0.685554	0.998784	0.598227
4	0.707676	0.999176	0.674703
5	0.669163	0.999022	0.638383
6	0.682788	0.998942	0.547073
7	0.640384	0.998818	0.577905
8	0.719236	0.998806	0.688446
9	0.713062	0.998237	0.706584
10	0.589428	0.998678	0.656742
11	0.653578	0.998859	0.589583
12	0.698739	0.999330	0.000000
13	0.699961	0.000000	0.000000
14	0.669479	0.998169	0.696686
15	0.670980	0.999086	0.536423
16	0.684841	0.998052	0.716760
17	0.688492	0.997968	0.716101
18	0.633417	0.998770	0.596831
19	0.688216	0.998819	0.617000
20	0.691495	0.999049	0.600317
21	0.713577	0.998710	0.650355
22	0.669259	0.999286	0.558268
23	0.688234	0.998807	0.576699
24	0.697139	0.999106	0.000000
25	0.646287	0.999044	0.454039
26	0.643229	0.999020	0.585591
27	0.685308	0.998491	0.640137
28	0.625459	0.000000	0.000000
29	0.702043	0.998537	0.692294
30	0.679788	0.997646	0.699128
31	0.713983	0.999027	0.654131
32	0.682019	0.000000	0.000000
33	0.621899	0.999176	0.496429
34	0.707237	0.998586	0.687039
35	0.688088	0.999025	0.526799
36	0.664517	0.998641	0.656568
37	0.719103	0.000000	0.000000
38	0.640995	0.999194	0.471519
39	0.687150	0.998842	0.641300
40	0.643748	0.000000	0.000000
41	0.697533	0.998963	0.620814
42	0.716745	0.998605	0.719456

```
43             0.617795      0.999344                 0.606705
44             0.572232      0.999456                 0.355678
45             0.687487      0.999225                 0.531682
46             0.668106      0.998971                 0.654814
47             0.672600      0.998475                 0.637427
48             0.669406      0.999214                 0.603574
49             0.627647      0.998892                 0.507840
```

```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume df contains your dataset

# Log transform skewed variables
skewed_vars = ['Total Cases', 'Total Deaths', 'Total Recovered',
'Active Cases', 'Total Tests']
df[skewed_vars] = np.log1p(df[skewed_vars])

# Select Features and Target Variable
selected_features = ['Total Deaths', 'Deaths/ 1M pop', 'Case Fatality
Rate']
X = df[selected_features]
y = df['Active Cases']

np.random.seed(42)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# Train Random Forest Regressor model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the model
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R^2 Score:", r2)

# Predict Active Cases for the next two years
# Assuming we want to predict for the next 24 months
# Assuming X_future contains features for the next 24 months (similar
to X)
```

```python
# Extracting values from existing DataFrame df
total_deaths_values = df['Total Deaths'].values
deaths_per_million_values = df['Deaths/ 1M pop'].values
case_fatality_rate_values = df['Case Fatality Rate'].values

# Generate synthetic data for the next 24 months
total_deaths_synthetic = [total_deaths_values[-1] + i * 50 for i in
range(1, 25)]  # Assuming an increase of 50 each month
deaths_per_million_synthetic = [deaths_per_million_values[-1] + i for
i in range(1, 25)]  # Assuming an increase of 1 each month
case_fatality_rate_synthetic = [case_fatality_rate_values[-1] + i *
0.002 for i in range(1, 25)]  # Assuming an increase of 0.002 each
month

# Create synthetic data dictionary
synthetic_data = {
    'Total Deaths': total_deaths_synthetic,
    'Deaths/ 1M pop': deaths_per_million_synthetic,
    'Case Fatality Rate': case_fatality_rate_synthetic
}
# Create X_future DataFrame
X_future = pd.DataFrame(synthetic_data)

# Use the trained machine learning model to predict active cases for
the next 24 months
predicted_active_cases = rf_model.predict(X_future)

# Visualize the predicted active cases for the next two years
months = range(1, 25)

# Convert predicted active cases from decimals to percentages
predicted_active_cases_percentage = predicted_active_cases * 100
plt.figure(figsize=(10, 6))
plt.plot(months, predicted_active_cases_percentage, marker='o',
linestyle='-')
plt.title('Predicted Active Cases for Next 24 Months')
plt.xlabel('Months')
plt.ylabel('Percentage of Predicted Active Cases')
plt.xticks(months)  # Set ticks for each month
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 0.0013268996327638776
R^2 Score: 0.09458241979800097
```




```
#Testing average mean cases for next 24 months

# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assume df contains your dataset

# Log transform skewed variables
skewed_vars = ['Total Cases', 'Total Deaths', 'Total Recovered',
'Active Cases', 'Total Tests']
df[skewed_vars] = np.log1p(df[skewed_vars])

# Select Features and Target Variable
selected_features = ['Total Deaths', 'Deaths/ 1M pop', 'Case Fatality
Rate']
X = df[selected_features]
y = df['Active Cases']

np.random.seed(42)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42)


# Train Random Forest Regressor model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate the model
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R^2 Score:", r2)

# Predict Active Cases for the next two years
# Assuming we want to predict for the next 24 months
# Assuming X_future contains features for the next 24 months (similar
to X)
# Extracting values from existing DataFrame df
total_deaths_values = df['Total Deaths'].values
deaths_per_million_values = df['Deaths/ 1M pop'].values
case_fatality_rate_values = df['Case Fatality Rate'].values

# Generate synthetic data for the next 24 months
total_deaths_synthetic = [total_deaths_values[-1] + i * 50 for i in
range(1, 25)]  # Assuming an increase of 50 each month
deaths_per_million_synthetic = [deaths_per_million_values[-1] + i for
i in range(1, 25)]  # Assuming an increase of 1 each month
case_fatality_rate_synthetic = [case_fatality_rate_values[-1] + i *
0.002 for i in range(1, 25)]  # Assuming an increase of 0.002 each
month

# Create synthetic data dictionary
synthetic_data = {
    'Total Deaths': total_deaths_synthetic,
    'Deaths/ 1M pop': deaths_per_million_synthetic,
    'Case Fatality Rate': case_fatality_rate_synthetic
}
# Create X_future DataFrame
X_future = pd.DataFrame(synthetic_data)

num_runs = 10
predicted_active_cases_list = []


for _ in range(num_runs):
    predicted_active_cases = rf_model.predict(X_future)
    predicted_active_cases_list.append(predicted_active_cases)

# Calculate the average predicted active cases
```

```python
average_predicted_active_cases = np.mean(predicted_active_cases_list,
axis=0)
average_predicted_active_cases_percentage = predicted_active_cases *
100

# Visualize the average predicted active cases for the next two years
months = range(1, 25)
plt.figure(figsize=(10, 6))
plt.plot(months, average_predicted_active_cases_percentage,
marker='o', linestyle='-')
plt.title('Average Predicted Active Cases for Next 24 Months (Average
of 10 runs)')
plt.xlabel('Months')
plt.ylabel('Percentage of Predicted Active Cases')
plt.xticks(months)  # Set ticks for each month
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 0.0006469154603926558
R^2 Score: 0.1035333581076553
```