```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('consolidated_ndvi_rain_cropland_FAO.csv')
```

```
df.tail()
```

Year	ndvi_data	rain_data(in mm)	Cases_Value	Deaths_Value	Province_Bomi	Province_Bong	Province_Gbarpolu	Province_Grand Bassa	Province_Grand Cape Mount	Province_Grand Gedeh	Province_Grand Kru	Province_Lofa	Province_
2016-01-01	0.571472	32.925278	63628.505842	167.958324	0	0	0	0	0	0	0	0	
2017-01-01	0.616583	36.737167	64100.238249	192.790002	0	0	0	0	0	0	0	0	
2018-01-01	0.590778	37.138889	63174.831790	199.265331	0	0	0	0	0	0	0	0	
2019-01-01	0.599743	40.547171	60955.889161	189.794016	0	0	0	0	0	0	0	0	
2020-01-01	0.576588	59.712647	58274.850530	232.665926	0	0	0	0	0	0	0	0	

Converting 'Year' column to datetime type

```
df['Year'] = pd.to_datetime(df['Year'], format='%Y')
```

```
df.set_index('Year', inplace=True)
```

Converting categorical variable 'Province' to numerical using one-hot encoding

```
df = pd.get_dummies(df, columns=['Province'])
```

```
df.head()
```

Year	ndvi_data	rain_data(in mm)	Cases_Value	Deaths_Value	Province_Bomi	Province_Bong	Province_Gbarpolu	Province_Grand Bassa	Province_Grand Cape Mount	Province_Grand Gedeh	Province_Grand Kru	Province_Lofa	Province_
2010-01-01	0.622944	29.284000	39421.904528	157.506441	1	0	0	0	0	0	0	0	
2011-01-01	0.631111	14.906528	39033.880960	155.825758	1	0	0	0	0	0	0	0	
2012-01-01	0.624000	22.661611	38417.442370	129.673104	1	0	0	0	0	0	0	0	
2013-01-01	0.629000	21.508250	40641.751749	119.172710	1	0	0	0	0	0	0	0	
2014-01-01	0.612833	28.455167	43748.805917	132.335863	1	0	0	0	0	0	0	0	

```
train_end_year = 2018
test_start_year = 2019
```

```
train_data = df[df.index.year <= train_end_year]
test_data = df[df.index.year >= test_start_year]
```

```
train_data.columns
```

```
Index(['ndvi_data', 'rain_data(in mm)', 'Cases_Value', 'Deaths_Value',
       'Province_Bomi', 'Province_Bong', 'Province_Gbarpolu',
       'Province_Grand Bassa', 'Province_Grand Cape Mount',
       'Province_Grand Gedeh', 'Province_Grand Kru', 'Province_Lofa',
       'Province_Margibi', 'Province_Maryland', 'Province_Montserrado',
       'Province_Nimba', 'Province_River Gee', 'Province_Rivercess',
       'Province_Sinoe'],
      dtype='object')
```

```
X_train = train_data.drop(['Cases_Value', 'Deaths_Value'], axis=1)
y_train_cases = train_data['Cases_Value']
y_train_deaths = train_data['Deaths_Value']

X_test = test_data.drop(['Cases_Value', 'Deaths_Value'], axis=1)
y_test_cases = test_data['Cases_Value']
y_test_deaths = test_data['Deaths_Value']
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
```

Model Training

Random Forest

```
rf_model_cases = RandomForestRegressor()
rf_model_cases.fit(X_train, y_train_cases)

rf_model_deaths = RandomForestRegressor()
rf_model_deaths.fit(X_train, y_train_deaths)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

XGBoost

```
[ ] xgb_model_cases = XGBRegressor()
    xgb_model_cases.fit(X_train, y_train_cases)

    xgb_model_deaths = XGBRegressor()
    xgb_model_deaths.fit(X_train, y_train_deaths)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

Model Evaluation

Random Forest evaluation

```
[ ] rf_preds_cases = rf_model_cases.predict(X_test)
    rf_rmse_cases = mean_squared_error(y_test_cases, rf_preds_cases, squared=False)

    rf_preds_deaths = rf_model_deaths.predict(X_test)
    rf_rmse_deaths = mean_squared_error(y_test_deaths, rf_preds_deaths, squared=False)

    print("Random Forest Cases RMSE:", rf_rmse_cases)
    print("Random Forest Deaths RMSE:", rf_rmse_deaths)
```

```
Random Forest Cases RMSE: 40586.84503911668
Random Forest Deaths RMSE: 87.57791048800657
```

```
    # XGBoost evaluation
    xgb_preds_cases = xgb_model_cases.predict(X_test)
    xgb_rmse_cases = mean_squared_error(y_test_cases, xgb_preds_cases, squared=False)

    xgb_preds_deaths = xgb_model_deaths.predict(X_test)
    xgb_rmse_deaths = mean_squared_error(y_test_deaths, xgb_preds_deaths, squared=False)

    print("XGBoost Cases RMSE:", xgb_rmse_cases)
    print("XGBoost Deaths RMSE:", xgb_rmse_deaths)
```

```
XGBoost Cases RMSE: 54509.45477169278
XGBoost Deaths RMSE: 109.09862384042488
```

Hyperparameter Tuning for Random Forest

```
[ ] from sklearn.model_selection import GridSearchCV
```

```
    param_grid_rf = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }

    grid_search_rf = GridSearchCV(estimator=RandomForestRegressor(), param_grid=param_grid_rf,
                                  cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

    grid_search_rf.fit(X_train, y_train_cases)

    #Best hyperparameters
    best_params_rf = grid_search_rf.best_params_

    # Training with the best hyperparameters
    best_rf_model_cases = RandomForestRegressor(**best_params_rf)
    best_rf_model_cases.fit(X_train, y_train_cases)

    # Evaluating again
    best_rf_preds_cases = best_rf_model_cases.predict(X_test)
    best_rf_rmse_cases = mean_squared_error(y_test_cases, best_rf_preds_cases, squared=False)

    print("Best Random Forest Cases RMSE:", best_rf_rmse_cases)
```

```
Best Random Forest Cases RMSE: 48024.808530311384
```

```
[ ] best_params_rf
```

```
{'max_depth': None,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 200}
```

```
    grid_search_rf_deaths = GridSearchCV(estimator=RandomForestRegressor(), param_grid=param_grid_rf,
                                         cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

    grid_search_rf_deaths.fit(X_train, y_train_deaths)

    best_params_rf_deaths = grid_search_rf_deaths.best_params_

    best_rf_model_deaths = RandomForestRegressor(**best_params_rf_deaths)
    best_rf_model_deaths.fit(X_train, y_train_deaths)

    best_rf_preds_deaths = best_rf_model_deaths.predict(X_test)
    best_rf_rmse_deaths = mean_squared_error(y_test_deaths, best_rf_preds_deaths, squared=False)

    print("Best Random Forest Deaths RMSE:", best_rf_rmse_deaths)
```

```
Best Random Forest Deaths RMSE: 101.38279057899965
```

```
[ ] best_params_rf_deaths
```

```
{'max_depth': 10,
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 100}
```

Hyperparameter Tuning for XGBoost

```
[ ] param_grid_xgb = {
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 5, 7],
        'learning_rate': [0.05, 0.1, 0.2],
        'subsample': [0.8, 0.9, 1.0],
        'colsample_bytree': [0.8, 0.9, 1.0]
    }

    grid_search_xgb = GridSearchCV(estimator=XGBRegressor(), param_grid=param_grid_xgb,
                                   cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

    grid_search_xgb.fit(X_train, y_train_cases)

    best_params_xgb = grid_search_xgb.best_params_

    best_xgb_model_cases = XGBRegressor(**best_params_xgb)
    best_xgb_model_cases.fit(X_train, y_train_cases)

    grid_search_xgb_deaths = GridSearchCV(estimator=XGBRegressor(), param_grid=param_grid_xgb,
                                          cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

    grid_search_xgb_deaths.fit(X_train, y_train_deaths)

    best_params_xgb_deaths = grid_search_xgb_deaths.best_params_

    best_xgb_model_deaths = XGBRegressor(**best_params_xgb_deaths)
    best_xgb_model_deaths.fit(X_train, y_train_deaths)

    best_xgb_preds_deaths = best_xgb_model_deaths.predict(X_test)
    best_xgb_rmse_deaths = mean_squared_error(y_test_deaths, best_xgb_preds_deaths, squared=False)

    print("Best XGBoost Deaths RMSE:", best_xgb_rmse_deaths)
    best_xgb_preds_cases = best_xgb_model_cases.predict(X_test)
    best_xgb_rmse_cases = mean_squared_error(y_test_cases, best_xgb_preds_cases, squared=False)

    print("Best XGBoost Cases RMSE:", best_xgb_rmse_cases)
```

```
Best XGBoost Cases RMSE: 57546.26620048158
```

```
[ ] best_params_xgb_deaths
```

```
{'colsample_bytree': 0.8,
 'learning_rate': 0.05,
 'max_depth': 7,
 'n_estimators': 300,
 'subsample': 1.0}
```

```
# Instantiate the GridSearchCV object
grid_search_xgb_deaths = GridSearchCV(estimator=XGBRegressor(), param_grid=param_grid_xgb,
                                      cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

grid_search_xgb_deaths.fit(X_train, y_train_deaths)

best_params_xgb_deaths = grid_search_xgb_deaths.best_params_

best_xgb_model_deaths = XGBRegressor(**best_params_xgb_deaths)
best_xgb_model_deaths.fit(X_train, y_train_deaths)

best_xgb_preds_deaths = best_xgb_model_deaths.predict(X_test)
best_xgb_rmse_deaths = mean_squared_error(y_test_deaths, best_xgb_preds_deaths, squared=False)

print("Best XGBoost Deaths RMSE:", best_xgb_rmse_deaths)
```

```
Best XGBoost Deaths RMSE: 123.3667335130727
```

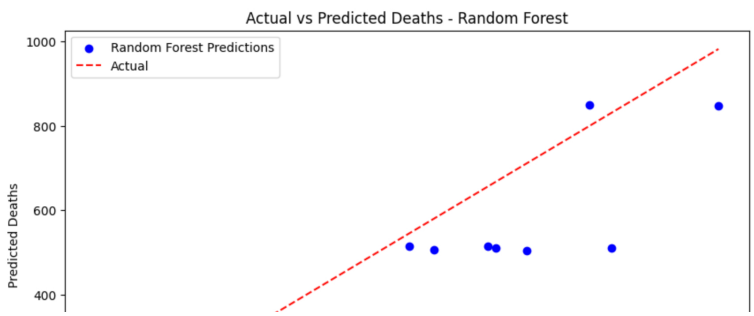
```
[ ] best_params_xgb_deaths
```

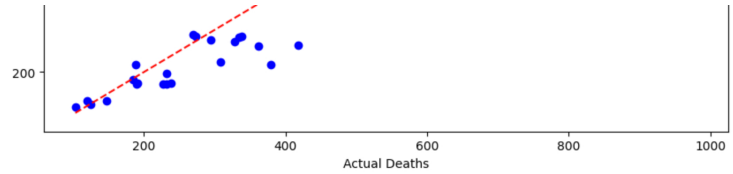
```
{'colsample_bytree': 0.8,
 'learning_rate': 0.05,
 'max_depth': 7,
 'n_estimators': 300,
 'subsample': 1.0}
```

Result Comparison RF Vs XGBoost

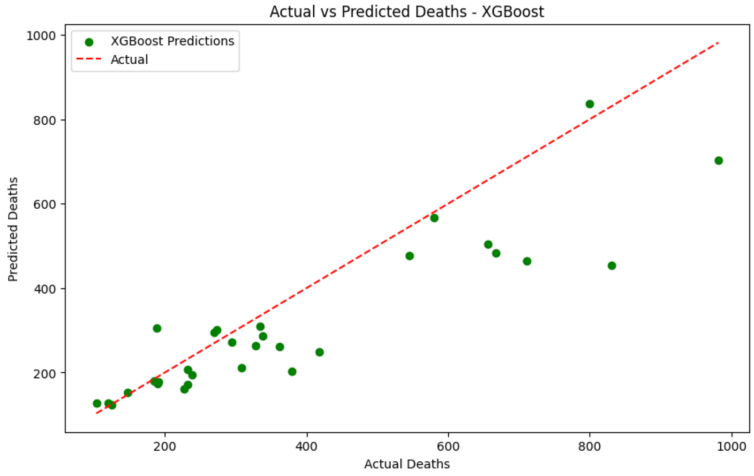
```
[ ] import matplotlib.pyplot as plt
```

```
[ ] plt.figure(figsize=(10, 6))
    plt.scatter(y_test_deaths, best_rf_preds_deaths, color='blue', label='Random Forest Predictions')
    plt.plot([min(y_test_deaths), max(y_test_deaths)], [min(y_test_deaths), max(y_test_deaths)], '--', color='red', label='Actual')
    plt.xlabel('Actual Deaths')
    plt.ylabel('Predicted Deaths')
    plt.title('Actual vs Predicted Deaths - Random Forest')
    plt.legend()
    plt.show()
```






```
[ ] plt.figure(figsize=(10, 6))
    plt.scatter(y_test_deaths, best_xgb_preds_deaths, color='green', label='XGBoost Predictions')
    plt.plot([min(y_test_deaths), max(y_test_deaths)], [min(y_test_deaths), max(y_test_deaths)], '--', color='red', label='Actual')
    plt.xlabel('Actual Deaths')
    plt.ylabel('Predicted Deaths')
    plt.title('Actual vs Predicted Deaths - XGBoost')
    plt.legend()
    plt.show()
```



Double-click (or enter) to edit

RMSE, MAE (Mean Absolute Error), and R-squared (coefficient of determination) scores

```
[ ] from sklearn.metrics import mean_absolute_error, r2_score
```

```
[ ] rf_rmse_cases = mean_squared_error(y_test_cases, best_rf_preds_cases, squared=False)
    rf_mae_cases = mean_absolute_error(y_test_cases, best_rf_preds_cases)
    rf_r2_cases = r2_score(y_test_cases, best_rf_preds_cases)

    rf_rmse_deaths = mean_squared_error(y_test_deaths, best_rf_preds_deaths, squared=False)
    rf_mae_deaths = mean_absolute_error(y_test_deaths, best_rf_preds_deaths)
    rf_r2_deaths = r2_score(y_test_deaths, best_rf_preds_deaths)
```

```
[ ] xgb_rmse_cases = mean_squared_error(y_test_cases, best_xgb_preds_cases, squared=False)
    xgb_mae_cases = mean_absolute_error(y_test_cases, best_xgb_preds_cases)
    xgb_r2_cases = r2_score(y_test_cases, best_xgb_preds_cases)

    xgb_rmse_deaths = mean_squared_error(y_test_deaths, best_xgb_preds_deaths, squared=False)
    xgb_mae_deaths = mean_absolute_error(y_test_deaths, best_xgb_preds_deaths)
    xgb_r2_deaths = r2_score(y_test_deaths, best_xgb_preds_deaths)
```

```
[ ] print("Random Forest Model Evaluation:")
    print("Cases RMSE:", rf_rmse_cases)
    print("Cases MAE:", rf_mae_cases)
    print("Cases R-squared:", rf_r2_cases)
    print("Deaths RMSE:", rf_rmse_deaths)
    print("Deaths MAE:", rf_mae_deaths)
    print("Deaths R-squared:", rf_r2_deaths)
```

```
Random Forest Model Evaluation:
Cases RMSE: 48024.808530311384
Cases MAE: 20572.837402733472
Cases R-squared: 0.8082582139794333
Deaths RMSE: 101.38279057899965
Deaths MAE: 71.84708888918706
Deaths R-squared: 0.8093340536846799
```

```
[ ] print("XGBoost Model Evaluation:")
    print("Cases RMSE:", xgb_rmse_cases)
    print("Cases MAE:", xgb_mae_cases)
    print("Cases R-squared:", xgb_r2_cases)
    print("Deaths RMSE:", xgb_rmse_deaths)
    print("Deaths MAE:", xgb_mae_deaths)
    print("Deaths R-squared:", xgb_r2_deaths)
```

```
XGBoost Model Evaluation:
Cases RMSE: 57546.26620048158
Cases MAE: 26804.0869984485
Cases R-squared: 0.7246913955317952
Deaths RMSE: 123.3667335130727
Deaths MAE: 83.24585926362765
Deaths R-squared: 0.7176805609053056
```

```
[ ] future_data = pd.DataFrame({
        'Year': [year for _ in range(len(future_provinces)) for year in future_years],
        'Province': future_provinces * len(future_years),
        'ndvi_data': future_ndvi_data * len(future_provinces),
        'rain_data(in mm)': future_rain_data * len(future_provinces)
    })
```

```
# Convert 'Year' column to datetime type
future_data['Year'] = pd.to_datetime(future_data['Year'], format='%Y')
# future_data.set_index('Year', inplace=True)
# Convert categorical variable 'Province' to numerical using one-hot encoding
future_data = pd.get_dummies(future_data, columns=['Province'])

expected_columns = ['ndvi_data', 'rain_data(in mm)',
                    'Province_Bomi', 'Province_Bong', 'Province_Gbarpolu',
                    'Province_Grand Bassa', 'Province_Grand Cape Mount',
                    'Province_Grand Gedeh', 'Province_Grand Kru', 'Province_Lofa',
                    'Province_Margibi', 'Province_Maryland', 'Province_Montserrado',
                    'Province_Nimba', 'Province_River Gee', 'Province_Rivercess',
                    'Province_Sinoe']

# Add missing columns
missing_columns = set(expected_columns) - set(future_data.columns)
for col in missing_columns:
    future_data[col] = 0

future_data = future_data[expected_columns]

print("Future Data:")
print(future_data)
```

```
Future Data:
    ndvi_data  rain_data(in mm)  Province_Bomi  Province_Bong  \
0        0.75               100              1              0
1        0.78               110              0              1
2        0.80                95              0              0
3        0.82               105              0              0
4        0.75               100              0              0
5        0.78               110              0              0
6        0.80                95              0              0
7        0.82               105              0              0
8        0.75               100              0              0
9        0.78               110              0              0
10       0.80                95              0              0
11       0.82               105              0              0
12       0.75               100              0              0
13       0.78               110              0              0
14       0.80                95              0              0
15       0.82               105              1              0
16       0.75               100              0              1
17       0.78               110              0              0
18       0.80                95              0              0
19       0.82               105              0              0
20       0.75               100              0              0
21       0.78               110              0              0
22       0.80                95              0              0
23       0.82               105              0              0
24       0.75               100              0              0
25       0.78               110              0              0
26       0.80                95              0              0
27       0.82               105              0              0
28       0.75               100              0              0
29       0.78               110              0              0
30       0.80                95              1              0
31       0.82               105              0              1
32       0.75               100              0              0
33       0.78               110              0              0
34       0.80                95              0              0
35       0.82               105              0              0
36       0.75               100              0              0
37       0.78               110              0              0
38       0.80                95              0              0
39       0.82               105              0              0
40       0.75               100              0              0
41       0.78               110              0              0
42       0.80                95              0              0
43       0.82               105              0              0
44       0.75               100              0              0
45       0.78               110              1              0
46       0.80                95              0              1
47       0.82               105              0              0
48       0.75               100              0              0
49       0.78               110              0              0
50       0.80                95              0              0
51       0.82               105              0              0
52       0.75               100              0              0
53       0.78               110              0              0
54       0.80                95              0              0
55       0.82               105              0              0
```

[ ]
```
# future_data = pd.DataFrame({
#     'Year': [year for _ in range(len(future_provinces)) for year in future_years],
#     'Province': future_provinces * len(future_years),
#     'ndvi_data': future_ndvi_data * len(future_provinces),
#     'rain_data(in mm)': future_rain_data * len(future_provinces)
# })
```

[ ]
```
future_data
```

29	0.78	110	0	0	0	0	0	0	0	0	0	0
30	0.80	95	1	0	0	0	0	0	0	0	0	0
31	0.82	105	0	1	0	0	0	0	0	0	0	0
32	0.75	100	0	0	1	0	0	0	0	0	0	0
33	0.78	110	0	0	0	1	0	0	0	0	0	0
34	0.80	95	0	0	0	0	1	0	0	0	0	0
35	0.82	105	0	0	0	0	0	1	0	0	0	0
36	0.75	100	0	0	0	0	0	0	1	0	0	0
37	0.78	110	0	0	0	0	0	0	0	1	0	0
38	0.80	95	0	0	0	0	0	0	0	0	1	0
39	0.82	105	0	0	0	0	0	0	0	0	0	1
40	0.75	100	0	0	0	0	0	0	0	0	0	0
41	0.78	110	0	0	0	0	0	0	0	0	0	0
42	0.80	95	0	0	0	0	0	0	0	0	0	0
43	0.82	105	0	0	0	0	0	0	0	0	0	0
44	0.75	100	0	0	0	0	0	0	0	0	0	0
45	0.78	110	1	0	0	0	0	0	0	0	0	0

46	0.80	95	0	1	0	0	0	0	0	0	0	0
47	0.82	105	0	0	1	0	0	0	0	0	0	0
48	0.75	100	0	0	0	1	0	0	0	0	0	0
49	0.78	110	0	0	0	0	1	0	0	0	0	0
50	0.80	95	0	0	0	0	0	1	0	0	0	0
51	0.82	105	0	0	0	0	0	0	1	0	0	0
52	0.75	100	0	0	0	0	0	0	0	1	0	0
53	0.78	110	0	0	0	0	0	0	0	0	1	0
54	0.80	95	0	0	0	0	0	0	0	0	0	1
55	0.82	105	0	0	0	0	0	0	0	0	0	0
56	0.75	100	0	0	0	0	0	0	0	0	0	0
57	0.78	110	0	0	0	0	0	0	0	0	0	0
58	0.80	95	0	0	0	0	0	0	0	0	0	0
59	0.82	105	0	0	0	0	0	0	0	0	0	0

Forecasting

Random Forest Forecasting

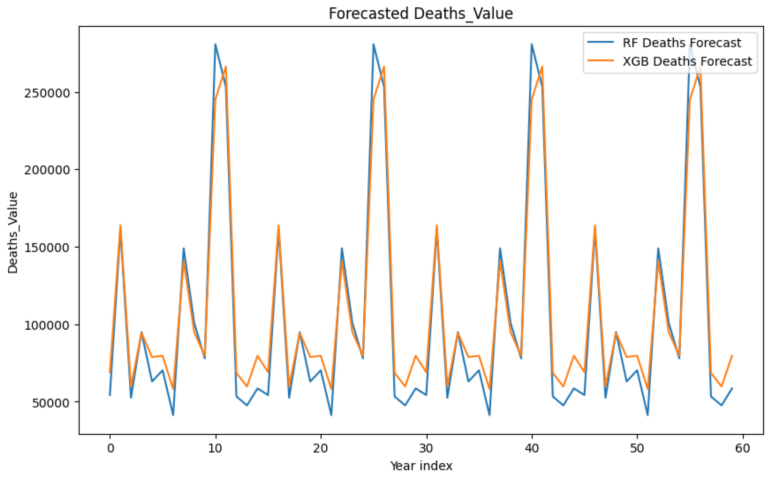
```
[ ] # Forecasting for Cases_Value
    future_cases_forecast_rf = best_rf_model_cases.predict(future_data)
```

XGboost

```
[ ] future_cases_forecast_xgb = best_xgb_model_cases.predict(future_data)
```

```
[ ] # Forecasting for Deaths_Value
    future_deaths_forecast_rf = best_rf_model_deaths.predict(future_data)
    future_deaths_forecast_xgb = best_xgb_model_deaths.predict(future_data)
```

```
[ ] plt.figure(figsize=(10, 6))
    plt.plot(future_data.index, future_cases_forecast_rf, label='RF Deaths Forecast')
    plt.plot(future_data.index, future_cases_forecast_xgb, label='XGB Deaths Forecast')
    plt.xlabel('Year index')
    plt.ylabel('Deaths_Value')
    plt.title('Forecasted Deaths_Value')
    plt.legend()
    plt.show()
```




```
[ ] plt.figure(figsize=(10, 6))
    plt.plot(future_data.index, future_deaths_forecast_rf, label='RF Deaths Forecast')
    plt.plot(future_data.index, future_deaths_forecast_xgb, label='XGB Deaths Forecast')
    plt.xlabel('Year index')
    plt.ylabel('Deaths_Value')
    plt.title('Forecasted Deaths_Value')
    plt.legend()
    plt.show()
```