Project Report: Predictive Modeling for Health Metrics Using Geographical Data

1. Introduction

This project aims to explore the predictive potential of geographical data (latitude and longitude) in forecasting critical health metrics during a hypothetical health crisis. The primary objectives are:

- **Predicting Deaths:** Estimating the number of fatalities in a given region using geographical and contextual data.
- **Predicting Case Fatality Ratio (CFR):** Estimating the proportion of confirmed cases that result in fatalities.

The project was executed in two phases:

1. **Phase 1:** Develop a model to predict the number of deaths (`Deaths`) using geographical features (`Lat`, `Long_`) and known case fatality ratio (`CFR`).
2. **Phase 2:** Build a multi-output model to predict both `Deaths` and `CFR` simultaneously, leveraging predictions from Phase 1 for enhanced accuracy.

This report provides an in-depth account of the methods, results, and insights obtained during the project.

---

2. Data Preparation

2.1 Data Loading and Inspection

- The dataset was loaded into a pandas DataFrame for preprocessing.
- Key initial inspections included:
    - Identifying missing values: `df.isnull().sum()`
    - Analyzing the data types and ensuring consistency across features.
    - Visualizing feature distributions with histograms to detect outliers or skewness.

2.2 Data Cleaning

- **Missing Values:**
    - Rows missing `Lat` or `Long_` values were removed to ensure the integrity of geographical information.

    - Cases missing `Deaths` but with valid `CFR` values below 1.0 were included in subset `df4` for potential inference in Phase 2.
    - A clean subset `df3` was created with complete `Deaths` and `CFR` information.
- **Validation:**
    - Ensured that `CFR` values were logical (0 ≤ `CFR` ≤ 1) and standardized to a decimal format (e.g., 20% → 0.2).

2.3 Data Transformation

- **Skewness Correction:**
    - Initial histograms revealed skewness in the distributions of `Deaths` and `CFR`.
    - A Yeo-Johnson transformation (`PowerTransformer`) was applied to normalize the data. Post-transformation distributions were re-visualized to confirm improvements in symmetry and normality.

---

3. Model Development

3.1 Phase 1: Predicting Deaths

Feature Engineering

- Selected features: `Lat`, `Long_`, and `CFR`.
- Explored potential feature enhancements, including:
    - Proximity to urban centers or healthcare facilities.
    - Regional clustering of data based on geographic proximity.

Preprocessing

- Data was split into training (80%) and testing (20%) sets using `train_test_split` with `random_state=42`.
- Features were standardized using `StandardScaler` to balance scale differences between latitude, longitude, and CFR.

Model Training

- **Model Choice:** XGBoost was selected for its robustness with non-linear relationships and ability to quantify feature importance.
- **Hyperparameter Tuning:**
    - Conducted using `RandomizedSearchCV` across parameters such as:
        - `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`.

    - Performed 5-fold cross-validation to evaluate model performance.

Evaluation Metrics

- The model was evaluated using:
    - Mean Absolute Error (MAE)
    - Mean Squared Error (MSE)
    - Root Mean Squared Error (RMSE)

Residual analysis was conducted to identify any systematic errors or areas for improvement.

Using Randomforest and RandomizedSearchCV for best parameters:

Optimized MAE: **34.91571464673521**

Optimized MSE: **1982.781626183728**

Optimized RMSE: **44.52843615246024**

Using XGBoost and RandomizedSearchCV for best parameters:

MAE: **34.711915946947904**

MSE: **1981.6247496249346**

RMSE: **44.515443945050514**

The missing deaths are stored in the df4 dataframe and saved for further training in phase 2.

3.2 Phase 2: Predicting Deaths and CFR Simultaneously

Data Preparation

- Predictions from Phase 1 (`Deaths`) were integrated into the dataset to replace missing values.

Model Training

- **Model Choice:** XGBoost with `MultiOutputRegressor` was used to handle the multi-output task of predicting both `Deaths` and `CFR`.
- **Hyperparameter Tuning:**
    - Conducted with `GridSearchCV` and `RandomizedSearchCV`, testing:
        - `n_estimators`, `max_depth`, `learning_rate`, and `subsample`.

    - 5-fold cross-validation ensured robust model selection.

Evaluation

- Metrics used for evaluation:
    - RMSE and MSE for both `Deaths` and `CFR`.

For XGBoost with GridSearchCV the results are:

Fitting 5 folds for each of 162 candidates, totalling 810 fits

Best Parameters: {'estimator__colsample_bytree': 1.0, 'estimator__learning_rate': 0.01, 'estimator__max_depth': 7, 'estimator__n_estimators': 300, 'estimator__subsample': 0.8}

Mean Squared Error (Deaths): 1482.620551366289

Root Mean Squared Error (Deaths): 38.50481205468076

Mean Squared Error (CFR): 4.6948837771375986e-05

Root Mean Squared Error (CFR): 0.006851922195367953

For Random Forest with RandomizedSearchCV the results are:

Fitting 5 folds for each of 50 candidates, totalling 250 fits

Best Hyperparameters: {'estimator__n_estimators': 100, 'estimator__min_samples_split': 2, 'estimator__min_samples_leaf': 4, 'estimator__max_features': 'log2', 'estimator__max_depth': None, 'estimator__bootstrap': True}

Mean Squared Error (Deaths) - Optimized Random Forest: 1436.309053540297

Root Mean Squared Error (Deaths) - Optimized Random Forest: 37.89866822911192

Mean Squared Error (CFR) - Optimized Random Forest: 4.313259949215887e-05

Root Mean Squared Error (CFR) - Optimized Random Forest: 0.006567541358237409

Clearly the **Random Forest with RandomizedSearchCV** performs better.

---

4. Results and Discussion

4.1 Model Performance

- **Phase 1:**

    - Achieved RMSE and MAE values (specific numerical values should be included based on results).
    - Feature importance analysis revealed the following hierarchy: `CFR` > `Lat` ≈ `Long_`.
- **Phase 2:**
    - RMSE and MSE values improved for both `Deaths` and `CFR` due to additional context from Phase 1 predictions.
    - The Randomforest-based multi-output model outperformed the Random Forest model in terms of accuracy and consistency.

4.2 Error Analysis

- Residual plots showed that prediction errors were higher in regions with extreme values of `Deaths` or `CFR`.
- Potential sources of error:
    - Lack of population density and healthcare facility data in the feature set.
    - Noise in the dataset due to missing or incomplete records.

4.3 Data Limitations

- Geographical data alone may not fully capture factors influencing health outcomes, such as:
    - Socioeconomic conditions.
    - Healthcare infrastructure and capacity.
    - Pandemic-specific variables (e.g., disease transmissibility).
- Missing data in the original dataset reduced the model's effectiveness in regions with sparse records.

---

5. Conclusion

- **Key Findings:**
    - Geographical data (latitude and longitude) combined with CFR offers moderate predictive capability for deaths in a health crisis.
    - Multi-output modeling with Randomforest significantly improves simultaneous prediction of `Deaths` and `CFR`.
- **Strengths:**
    - Robust preprocessing and feature selection improved model reliability.
    - Use of hyperparameter tuning and cross-validation ensured generalizable performance.
- **Limitations:**

- The lack of auxiliary data (e.g., population density) limited the model's full potential.
- Dependence on accurate CFR values for regions with missing death records introduced potential bias.

Explanations of codes:

Here's a more visually appealing representation of your code steps with actions and the code:

Data Preparation

1. Reading Data:

- **Action:** Loaded the dataset from a CSV file into a pandas DataFrame.

```
df = pd.read_csv('/Users/harshikantdubey/Documents/ALL_ /train_data.xlsx - Sheet1.csv')
```

2. Checking Missing Values:

- **Action:** Examined the dataset for any missing entries across columns.

```
df.isnull().sum()
```

3. Removing Rows with Missing Lat or Long Values:

- **Action:** Removed rows where 'Lat' or 'Long_' data was missing to maintain geographical accuracy.

```
df1 = df.dropna(subset=['Lat', 'Long_'])
```

4. Handling Missing 'Deaths' and 'Case_Fatality_Ratio':

- **Action:** Filtered and processed rows where 'Deaths' were missing but 'Case_Fatality_Ratio' was valid.

```
df4 = df1[df1['Deaths'].isna()]
df4 = df4[df4['Case_Fatality_Ratio'] <= 100]
df4.rename(columns={'Case_Fatality_Ratio': 'CFR'}, inplace=True)
df4['CFR'] = df4['CFR'] / 100
```

5. Cleaning 'Deaths' and 'Case_Fatality_Ratio':

- **Action:** Further cleaned the dataset by removing rows with missing values for 'Deaths' or 'CFR', ensuring validity.

```
df2 = df1.dropna(subset=['Deaths'])
df3 = df2.dropna(subset=['Case_Fatality_Ratio'])
```

```
df3 = df3[df3['Case_Fatality_Ratio'] <= 100]
df3.rename(columns={'Case_Fatality_Ratio': 'CFR'}, inplace=True)
df3['CFR'] = df3['CFR'] / 100
```

6. Saving Cleaned Data:

- **Action:** Saved the processed DataFrame for future use or analysis.

```
df3.to_csv('/Users/harshikantdubey/Documents/ALL_ /train_data_accurate_.csv',
index=False)
```

Model Building (Random Forest and XGBoost)

7. Feature and Target Definition:

- **Action:** Defined features (inputs) and the target (output) for the model.

```
X = df3[['Lat', 'Long_','CFR']]
y = df3['Deaths']
```

8. Data Splitting and Scaling:

- **Action:** Divided the data into training and testing sets, then scaled the features to normalize input.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

9. Model Training and Evaluation (Random Forest):

- **Action:** Trained a Random Forest model on the scaled data.

```
random_forest = RandomForestRegressor(random_state=42, n_estimators=100)
random_forest.fit(X_train_scaled, y_train)
```

10. Hyperparameter Tuning (Random Search):

- **Action:** Optimized the Random Forest model using RandomizedSearchCV to find better hyperparameters.

```
random_search =
RandomizedSearchCV(estimator=RandomForestRegressor(random_state=42), ...)
best_params = random_search.best_params_
```

11. Model Training with XGBoost:

- **Action:** Trained an XGBoost model, potentially for better performance.

```
xgb_model = xgb.XGBRegressor(...)
xgb_model.fit(X_train_scaled, y_train)
```

Prediction on New Data

12. Prediction with Loaded Model:

- **Action:** Used the trained model to predict deaths for new data points after scaling.

```
new_data_scaled = scaler.transform(new_data)
predicted_deaths = loaded_model.predict(new_data_scaled)
```

Combining DataFrames

13. Combining Cleaned Data:

- **Action:** Merged datasets to include both actual and predicted data.

```
combined_df = pd.concat([df3_reset, df4_reset], ignore_index=True)
```

Final Data Handling

14. Data Transformation for Skewness:

- **Action:** Applied a power transformation to normalize 'Deaths' and 'CFR' distributions.

```
transformer = PowerTransformer(method='yeo-johnson', standardize=True)
data[['Deaths', 'CFR']] = transformer.fit_transform(data[['Deaths', 'CFR']])
```

15. Multi-output Model:

- **Action:** Trained a model to predict both 'Deaths' and 'CFR' simultaneously.

```
multioutput_model = MultiOutputRegressor(xgb_model)
multioutput_model.fit(X_train_scaled, y_train)
```

16. Hyperparameter Tuning and Prediction on Test Data:

- **Action:** Continued tuning and evaluating models for better performance.

Multi-Output Model for Deaths and CFR

15. Multi-Output Model Setup:

- **Action:** Prepared data for dual prediction of 'Deaths' and 'CFR'.

```
X = combined_df[['Lat', 'Long_']]
y = combined_df[['Deaths', 'CFR']]
```

16. Data Splitting and Scaling:

- **Action:** Split and scaled the data for multi-output prediction.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

17. XGBoost Model Training with MultiOutput:

- **Action:** Trained an XGBoost model for simultaneous prediction of 'Deaths' and 'CFR'.

```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', eval_metric='rmse',
random_state=42)
multioutput_model = MultiOutputRegressor(xgb_model)
multioutput_model.fit(X_train_scaled, y_train)
```

18. Model Evaluation:

- **Action:** Assessed model performance using error metrics like MSE and RMSE.

```
y_pred = multioutput_model.predict(X_test_scaled)
mse_deaths = mean_squared_error(y_test['Deaths'], y_pred[:, 0])
rmse_deaths = np.sqrt(mse_deaths)
mse_cfr = mean_squared_error(y_test['CFR'], y_pred[:, 1])
rmse_cfr = np.sqrt(mse_cfr)
```

19. Hyperparameter Tuning with GridSearchCV:

- **Action:** Fine-tuned the XGBoost model parameters using GridSearchCV.

```
param_grid = {...}
grid_search = GridSearchCV(estimator=multioutput_model, param_grid=param_grid, ...)
grid_search.fit(X_train_scaled, y_train)
```

20. Random Forest Model with RandomizedSearchCV:

- **Action:** Explored Random Forest for multi-output prediction with randomized tuning.

```
rf_model = RandomForestRegressor(random_state=42)
multioutput_model_rf = MultiOutputRegressor(rf_model)
random_search = RandomizedSearchCV(estimator=multioutput_model_rf, ...)
random_search.fit(X_train_scaled, y_train)
```

Prediction and Analysis of Total Cases

21. Loading and Preparing Test Data:

- **Action:** Loaded and cleaned new test data for predictions.

```
df6 = pd.read_csv('/Users/harshikantdubey/Documents/ALL_ /test_points.xlsx - Sheet1.csv')
df6 = df6.dropna(subset=['Lat','Long_'])
```

22. Prediction on New Data:

- **Action:** Made predictions on new geographical points.

```
X_df6_scaled = scaler.transform(X_df6)
y_pred = best_model_rf.predict(X_df6_scaled)
```

23. Adding Predictions to DataFrame:

- **Action:** Added the predictions to the DataFrame for further analysis.

```
df7['Predicted_Deaths'] = predictions_df['Deaths']
df7['Predicted_CFR'] = predictions_df['CFR']
```

24. Calculating Total Cases:

- **Action:** Estimated total cases from predicted deaths and CFR.

```
combined_df['Predicted_Total_Cases'] = combined_df.apply(lambda row:
row['Predicted_Deaths'] / row['Predicted_CFR'] if row['Predicted_CFR'] != 0 else np.nan,
axis=1)
```

25. Handling NaN Values:

- **Action:** Managed missing values in total case predictions by setting them to zero.

```
combined_df['Predicted_Total_Cases'].fillna(0, inplace=True)
```